ESD-TR-71-81

OPERATING SYSTEM VALIDATION TESTING

William C. Mittwede
Kenneth P. Choate

January 1971

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

(Prepared under Contract No. F19628-70-C-0258 by The COMTRE Corporation, 151 Sevilla Avenue, Coral Gables, Florida 33134.)

LEGAL NOTICE

When U. S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

ESD-TR-71-81

OPERATING SYSTEM VALIDATION TESTING

William C. Mittwede
Kenneth P. Choate

January 1971

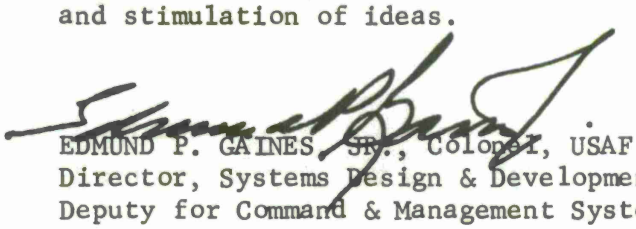
DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

(Prepared under Contract No. F19628-70-C-0258 by The COMTRE Corporation, 151 Sevilla Avenue, Coral Gables, Florida 33134.)

FOREWORD

This report presents the results of an analysis conducted by The COMTRE Corporation of Coral Gables, Florida, in support of Project 6917, Task 691701 under Contract F19628-70-C-0258. The analysis presented in this report was performed by William C. Mittwede and Kenneth P. Choate. Dr. John B. Goodenough (ESD/MCDS) was the ESD Project Monitor.

Publication of this report does not constitute Air Force approval of the report's findings or conclusions. It is published only for the exchange and stimulation of ideas.



EDMUND P. GAINES, SR., Colonel, USAF
Director, Systems Design & Development
Deputy for Command & Management Systems

ABSTRACT

This report presents functional testing requirements for use in the validation testing of computer operating systems. The requirements are structured in a tabular format and are applicable to the executive/control functions, system management functions and data manipulation functions of current commercially available operating systems. In concert with the tabulation of requirements for each of the operating system functions, further tabulation has also been performed relating the test requirements to the type of environment that the operating system must support: batch, real-time, or time-sharing. Basic testing procedures have been defined to verify the requirements and these testing methods have then been grouped into test packages.

CONTENTS

SECTION I
INTRODUCTION

1.1 Purpose

This is the third report of a series produced by The COMTRE Corporation for the Electronic Systems Division of the Air Force Systems Command. The first report of this series, ESD-TR-70-377, presented an integrated functional classification structure applicable to the executive/control functions, system management functions, and data manipulation functions of current commercially available operating systems. The second report developed selection criteria and the methods for establishing a relationship between these criteria and the operational requirements derived from the functions given in the first report.

In this report, validation requirements have been developed within the functional classification scheme for all levels and types of operating systems supporting current computer configurations. These validation requirements are presented in tabular form to allow easy selection of pertinent tests based upon fundamental applications of the Operating System (OS) in question.

The objective of this report is threefold:

1) to assure a high degree of completeness as well as uniformity in OS acceptance test design;
2) to increase the utility of benchmark programs currently used to debug, test, and validate operating systems;
3) to identify facilities that can be inserted into operating systems which will assist in the validation process.

The analysis presented within this report is based upon the assumption that validation should have two distinct objectives:

- to verify the presence of OS functions and their proper performance; and
- to observe the effects of benchmarks to assure that the system does perform as a unit.

Benchmark programs were designed by considering all OS functions as outlined by the functional classification scheme, determining the types of tests necessary to validate the functions, and then organizing these tests into a logical series of test packages applicable to varying system orientations.

The approach taken to the area of OS-assisted validation was to survey current techniques in the field of system measurement and to extrapolate relevant techniques for system validation. Also, several existing and proposed debugging techniques were investigated for possible application to validation.

1.2 Scope

This report defines procedures, software and data for systematic validation testing of current commercially available operating systems. The validation testing requirements presented encompass all significant levels and types of operating systems and are structured to permit selection and generation of tests for any given operating system.

The report is organized into five sections with two supporting appendices. The next Section presents the current approaches to OS measurement, their limitations, and the conclusions reached by this analysis. Section 3 is a concept development of system-assisted testing and Section 4 is a delineation of the OS functional testing requirements. Section 5 is a test design for validation. The appendices comprise a bibliography of documents reviewed during the study and a record of interviews with various representatives on the technical aspects of OS validation.

SECTION II

OPERATING SYSTEM MEASUREMENT

2.1 Current Approaches

The performance of operating system measurement has long been an area of interest within the computer community and is even more important with the advent of third generation operating systems which often support multiple processing configurations in addition to multiprogramming capabilities. In many instances system measurement and system validation are construed to be one and the same thing. Although these two functions can be complementary, they are, in fact, quite different. Measurement may be considered a system design tool to ascertain that the best possible performance is being delivered by a system or to determine why a system is not performing properly. Validation, on the other hand, is a process for determining if a system's performance is within the requirement specifications for a given facility. In other words, system measurement techniques are used to "tune" an operating system in an attempt to achieve its ultimate capabilities and operating system validation determines if the designed system satisfies a facility's specific requirements. This does not mean that system measurement techniques would not be useful to a facility in evaluating prospective systems or for further improvement or testing of its system after acceptance; however, system measurement techniques are most appropriately applied during operating system design. Since it is possible that certain measurement techniques may be applicable to operating system validation, a survey has been performed of some of the current measurement methods.

Currently, there are two classes of measurement techniques that have been developed for acquiring operating system measurements. These are the hardware instrumentation methods which involve the attachment of "probes" to various computer components to record activity in certain areas of the computer system and the software instrumentation method which involves either a modification to the basic operating system or the addition of software routines which have access to pertinent areas within the operating system.

In surveying the hardware monitoring devices available today, it appears that they all provide nearly the same type of measurements. The basic factors provided are CPU utilization, I/O channel utilization and peripheral device utilization. Other measurements usually provided include system overhead, data base activity, allocation of time between problem programs and the operating system, op code usage, time within a given memory

area, number of entries/exits from a given routine, operator response time, number of instructions executed, etc. Usually, this type of information is then processed and presented in report form indicating the system's performance activity and utilization.

As is evidenced by the type of measurements provided by the hardware monitoring devices, the major goal is improving system efficiency or, in evaluation, determining which system is the most efficient.

The major advantage in using hardware monitoring methods over software monitoring methods is that the former does not introduce any overhead into the system and therefore a true operational environment is measured.

A typical hardware monitoring device is the X-RAY system manufactured by the Applied Systems Division of the Computer Learning and Systems Corporation. A general description of this system as stated in the X-RAY system manual is as follows: X-RAY is designed to measure the total efficiency of a computer system. X-RAY provides a data collection, reduction and analysis facility for accurate reporting of all aspects of system performance including such major areas of interest as:

1. Computer System Utilization;
2. System Program Overhead;
3. Problem Program Efficiency;
4. Data Base Element Activity.

X-RAY isolates specific areas of operating inefficiency so that system improvement measures such as the following may be applied:

1. Equipment Configuration Balancing;
2. Job Scheduling Procedure Modification;
3. Operating System Residence Reallocation;
4. Program Structure and Code Optimization;
5. Data Base Reorganization/Redistribution.

The X-RAY recorder samples hardware registers, indicators and lines using a passive signal acquisition technique. The data samples are recorded on magnetic tape for post-processing by a software package designated as the X-RAY/Analyzer which produces reports describing configuration usage, program execution and data base activity. The X-RAY/Analyzer also provides a facility with the capability to generate its own reports.

Other hardware monitoring devices currently available are:

- The Computer Performance Monitor II (CPM II) which is produced by Allied Computer Technology, Inc. This system is designed to locate system imbalance and monitor system utilization as a tool in evaluation of system operation and program performance.

- The CPA 7700, produced by Computer Programming and Analysis, Inc., which provides measurements of such items as system wait state, I/O utilization, and problem and supervisor time allocation.

- The System Utilization Monitor (SUM) which is produced by Computer Synectics, Inc. This system measures: wait state time, channel usage, operator response time, problem time versus supervisor time, seeks performed, number of instructions executed, number of cards read/punched, I/O errors, CPU errors, storage errors, etc.

- The Dynaprobe system, produced by COMRESS, which is designed basically to study central processor activity versus data channel and I/O device usage. This system provides measurements such as system state, CPU active time, CPU wait time, system idle time, system active time, channel busy time, device idle time, instruction class, file access rate, etc.

In surveying the software measurement systems available today it appears that they provide the same types of information as provided by the hardware monitoring systems, with one important exception. This exception is the fact that the software measurement systems provide the capability of presenting the causal element. In other words, that program which caused an event to occur can be determined. This factor is very important in answering utilization questions during system monitoring and is a definite aid in "tuning" a system.

The software measurement systems surveyed were:

- Boole and Babbage, Systems Measurement Software (SMS/360), Configuration Utilization Evaluator (MCUE, Version 1) (CUE, Version 2). This system extracts and analyzes data describing hardware usage, data-cell or disk head movements, transient supervisor call routine loading, etc.

- Boothe Resources International, Inc., Computer Installation Management System (CIMS/I). This system records and analyzes data describing job step nomenclature, CPU utilization, hardware usage, I/O requests, etc.
- Webster Computer Corporation, S/360 Disk Operating System, Machine Utilization Reporting System (DOS MURS). This system extracts and analyzes data describing program nomenclature, CPU utilization, core utilization, I/O wait time, operator ID, etc.
- Computing Efficiency, Inc., COMPUMETER. This system extracts and analyzes data regarding the utilization and cost related to the computer system, the computer operators, and the programming staff.
- Computer Learning and Systems Corporation, Computer-Aided System Evaluation (CASE). This system utilizes simulation to determine file requirements, file utilization, and input component utilization.

Two measurement systems of interest which have been developed and utilized are the Data Collection Facility (DCF), presented by T. B. Pinkerton (see reference 13 in the bibliography) and the instrumentation methods utilized in the measurement of Multics (see reference 15 in the bibliography).

The DCF was developed as a monitoring system for a time-sharing system. The monitoring system itself was designed into the operating system providing information which is more detailed than data sampling methods but not approaching hardware monitor resolution. The interesting facet of this system is the minimal amount of interference that is introduced into the system by the DCF (this factor can affect system assisted validation). Also, the conclusions reached during the research performed in support of the development of DCF are very important and should be considered in the design of any software performance measurement system. These conclusions are:

- attempt to associate overhead caused by measurement with processes independent of those being measured;
- defer analysis of data for post processing;
- provide capabilities to choose among data to be extracted;
- attempt to utilize continuous data extraction rather than sampling;

- provide a monitoring system which is an integral part of the operating system and can be used during normal operation.

The measurement of multics is interesting in light of the development of measurement techniques as an integral part of the system design. Also, the utilization of hardware devices in conjunction with software and integrated into the system provides for the utilization of the best features of both types of measurement. The instrumentation of Multics was utilized during system design and was directed primarily toward an understanding of the internal operation of the operating system rather than measuring throughput, system capacity, or the characteristics of the system load. However, areas of interest which are directly applicable to software validation testing are the tracing package and simulation script utilized by the system. The tracing package performs a continuous looping function in which the calendar clock is continuously read. Normally successive clock readings will vary by the loop transit time, large differences are caused by control being given to another process. By analyzing the output of these recorded differences and by utilizing a known operational scenario it would be possible to validate system interrupt handling, algorithmic scheduling and proper peripheral utilization. The simulation script is much like a known benchmark program in that it offers a known measurable operational scenario. It is important in system validation testing that a known or controlled operational scenario be utilized and the simulation script method offers this type of scenario.

2.2 Limitations

The major limitation uncovered in the survey of commercially available software measurement systems is their dependency upon the operating system. This is caused by the measurement system's requirement to extract necessary information from core. This extraction is dependent upon timing parameters and format of information which is a function of the individual operating system. Therefore, each measurement system is written to perform with a particular operating system. Thus, for each operating system, a unique measurement system is required. It was found that the majority of software measurement systems only provide their service for the IBM 360 system. Also, software measurement systems tend to introduce a certain amount of overhead into an operating system. Although it is usually stated that this is usually stated that this is a minimum factor, it must be considered nevertheless.

The hardware measurement systems surveyed offer the same type of information as the software measurement systems and are able to perform this function without introducing overhead into the operating system. However, the hardware measurement systems lack the flexibility of their software counterparts and, although they can present utilization factors, they cannot present the utilization causal factors which is possible using software measurement systems.

2.3 Conclusions

From the material surveyed on commercially available measurement systems, it appears that these tools are truly useful to a facility in "tuning" an operating system to best satisfy its operational requirements and that they can provide a means for obtaining more efficient utilization of the system. It is quite plausible to consider that ultimately system measurement will be performed by a hybrid system, encompassing the best features of hardware and software measurement systems. The problem of requiring a different measurement system for each operating system has no apparent solution because of the inherent differences among operating systems. This will continue to be the case until standardized measurement recording requirements are imbedded within each operating system during design.

The simulation tool surveyed appears to be a highly useful device for a facility attempting to determine the hardware/software configuration that best satisfies its requirements but does not apply to system measurement as performed by the systems surveyed.

The type of information obtained by both the hardware and software measurement systems can be very useful in evaluating the capabilities of different systems when used in conjunction with a standard benchmark program. However, these measurements appear to have little significance in the validation of operating systems with the exception of determining which system best utilizes its resources and which system could best satisfy peak loading conditions.

SECTION III
SYSTEM-ASSISTED TESTING

3.1 Concept

Many of the services afforded an application program by the system supervisor are rather easily validated by simple test programs. For example, the capability to issue I/O commands, request the time of day, take a core dump, etc., can be tested in a fairly straightforward manner. The difficult system area to validate is supervisory and management control. For example, the areas of dynamic allocation, multiprogramming control (scheduling and dispatching), job and task management, total system management, etc., are representative of those functions that cannot be directly observed. Insofar as a test program is usually a single application program operating independently of any other application program, it is difficult, except in trivial cases, to develop the timing inter-relationships which cause the system to exercise its supervisory control functions.

For this reason, this section presents several approaches which, if implemented, will allow the behavior of the supervisor to be observed in a manner that is not now commonly possible. Each of these approaches involves additions and modifications to the design of existing supervisor programs. Some of these capabilities are fairly easily provided with the addition of a minimal amount of coding; others may, in some supervisory structures, require extensive program re-design. It is felt, however, that these modifications will permit a level of system validation that has not been previously possible. A further benefit, though of somewhat less importance than the validation aspect, is that the ability to observe a supervisor's control operation will also facilitate debugging operations when supervisor errors are encountered.

The concept, called system assisted-testing, is based upon the inclusion within the supervisor of a number of routines which record various system actions for immediate or subsequent visual verification, other routines which create conditions to which the control program must respond, and selected facilities which enable certain system control variables to be dynamically modified during supervisor validation proceedings.

Since validation procedures are normally conducted quite independently of normal system operation, each of the procedures mentioned should not be permanently installed within the operational supervisor. Rather, the concept is based upon a special mode of operation called, perhaps, the validation mode, wherein the supervisor will be dynamically augmented by the addition of the validation routines.

Thus, if the validation mode is specified during system initialization the supervisor nucleus would be modified to enable linkages to the actual validation routines. These routines, depending upon the design of the particular system, could then be either loaded as a part of the system nucleus, established in a privileged supervisor partition, or called dynamically into a transitional area when referenced.

A disadvantage to this approach is that the supervisor being validated is somewhat modified from the actual operational supervisor. However, this disadvantage is compensated for by the fact that the resident operational supervisor will have a smaller main storage requirement and/or a somewhat faster mode of operation when validation techniques are inactive. Since the frequency of validation procedures is quite small compared to normal operating time, the tradeoff of time and core seems justified.

The following Subsections present four system-assisted validation techniques. Each attempts to provide a slightly different technique and is independent of the others. Thus, any or all might be incorporated into an existing or proposed system depending upon the level and type of validation desired.

3.2 Event Logging for Post-Mortem Analysis

The first technique is based upon a capability implemented by the designers of the General Electric Comprehensive Operating System III (GECOS III - see Reference 5 in the Bibliography). To the authors' knowledge, this technique was not available in operational versions of the system, but was, instead, incorporated into pre-production testing of the system.

The capability should be invoked by a system control card or an operator key-in during system initialization. Invocation enables a system trace or logging routine which will record the occurrence of various events upon a log file on a dedicated output device. A fairly high blocking factor should also be provided to reduce I/O interference. The log file is available for subsequent analysis by a series of general

purpose routines which will reduce the data collected to a series of charts depicting an overview of system operation.

Most of the event logging routines (I/O buffers, trace file write, initialization) need not be imbedded within the supervisor nucleus. Rather, they can be loaded into a permanent area for execution when the capability is activated. The required changes in the supervisor nucleus are fairly minimal and should not significantly alter the nucleus size.

Operation: When initialized for validation, each time a specified event occurs, an event record will be constructed and transferred to the system log file. The event record should look somewhat like the following:

TIME OF DAY	EVENT CODE	JOB OR SYSTEM REFERENCE NUMBER	TASK NUMBER	INTERRUPT OR ERROR CODE	RESOURCE ID

The following events are indicative of the conditions that should cause the production of a logging record:

01 Recognition of a new job submitted to the system

02 Placing a new job on the scheduling queue

03 Removing a job from the scheduling queue and placing it in the executing job mix

04 Removing a job from the scheduling queue for another reason (e.g., operator command)

05 Initiating a task within a job

06 Assigning a single resource to a job or task (allocation)

07 Releasing a single resource by a job or task (de-allocation)

08 Removing a single resource from a job or task (operator-directed action)

09 Assigning the CPU to a task

10 Removing the CPU from a task

11 Loading a program page or segment

12 Releasing or overlaying a program page or segment

13 Rolling out a program area

14 Rolling in a program area

15 Initiating core compaction

16 Normal task termination

17 Abnormal task termination

18 Job termination

19 Interrupt occurrence (A certain selectivity should be provided for the types of interrupts logged. For example, I/O interrupts might well be excluded.)

20 Exceeding a pre-specified program limit (core space, time, records, etc.)

21 Hardware error occurrence

22 Program error occurrence

23 Recognition of a new system resource

24 Deletion of an existing system resource

25 Receipt of a computer operator command relating to a resource or job

26 Initiation of symbiont routine

27 Termination of symbiont routine

28 Start of output symbiont processing for a specific job

29 End of output symbiont processing for a specific job

Validation technique: A pre-planned scenario of system benchmarks should be prepared to simulate normal system operation. Only those system events relating to specific test objectives should be activated; the others should remain dormant. Once the log file has been obtained, it will be processed by one or more data reduction programs to produce a map of the internal system actions of interest. This map, in turn, can be visually validated to assure that the event sequences correspond to the actual steps the system is required to perform.

The following layouts are examples of the types of data reduction maps that could be produced to validate various operating system functions:

1. Scheduling Process

   For each job scheduled:

   time of day the job was scheduled

   other jobs remaining in the job scheduling queue

   length of time each job has been in the scheduling queue

   any resource assignments that have been made to unscheduled jobs

2. Time Slicing or Priority Dispatching Algorithm

   job/task name

   time processor assigned to the task

   time processor removed from the task

3. Degree of Multiprogramming or Time-Sharing

   For each job scheduled or terminated:

   time of day

   job initiation or job termination indicator

   job name

   number of jobs/tasks currently active in the job mix

4. Peripheral Device Allocation

   For each peripheral device:

   time assigned to a specific job/task

   time released by a job/task

   job/task name

   If dynamic allocation is available, then the following information should also be included:

   time of job/task initiation,

   time of job/task termination.

   If the device is added or deleted from the system, then an appropriate message should also be included.

5. Memory Management

a) Paged memory environment

For each job:

time of day

identification of page loaded or removed

indicator for pages loaded: was previous page swapped out prior to loading?

number of pages currently active for the job

b) Non-paged environments

For each instance of storage compaction:

time of day

previous memory map

new memory map

For each instance of program roll out:

time of day

program (name) rolled-out

program (name) causing roll out

previous memory map

new memory map

For each job using an overlay structure:

initial core storage assignment

any modifications to core storage assignments with the corresponding time of day

For each instance of overlay:

time of day

core area overlayed

name of overlaying segment

6. Symbiont Processing

a) Input symbionts

time symbiont is initiated

time symbiont is terminated

For each job:

time job is initially recognized by the system

time job is placed on the scheduling queue

time job is entered into job mix

b) Output Symbionts

time symbiont is initiated

time symbiont is terminated

For each job:

time job is terminated

time symbiont processing is initiated for the job

time symbiont processing is completed for the job

7. Task Sequencing/Program Termination Control

For each job, list the following events and the respective time of event occurrence:

job initiation

task initiation

task termination

job termination

device allocation

device de-allocation

program error occurrences

hardware error occurrences

program limits exceeded

8. Hardware Error Control

This display should list all non-scheduling related events that occur from the time of error recognition until a new job is selected from the scheduling queue. This should produce a trace of all interactions that might occur due to unrecoverable errors (e.g., re-allocation of resources, suspension of intermediate processing, etc.).

The event logging facility thus imposes a minimal impact upon the system undergoing validation and allows a rather comprehensive post-mortem analysis of the system control functions being tested. The suggested examples of data reduction are by no means complete. Rather, they are indicative of the varying types of functional verification that may be provided. A comprehensive test employing this technique is limited only by the ingenuity of the test designer and the extensiveness of the event logging facility. Furthermore, this approach may be extended to validate new functional capabilities incorporated into future operating systems.

3.3 Event Simulation

One of the major problems in validating an operating system is to create a series of time-related events to which the system must react. Insofar as the system is proceeding at a rate measured in nano- or microseconds, it becomes virtually impossible for a human operator to cause specific events to occur within selected time constraints. At best, he can provide events at a tolerance measured in seconds. Furthermore, a comprehensive test of a time-sharing system, for example, requires the close coordination of a number of remotely located terminal operators which only further compounds the problem.

The technique of event simulation has been used quite successfully to test and validate special purpose real-time and/or time-sharing features. In particular, the event simulator provides an almost unique capability to test system overload and other time-dependent relationships.

To prepare for event simulation, the test designer creates a time-dependent scenario of system events. The types of events to be considered are those that are external to the system; activation (log-on) of a local or remote terminal, arrival of an input message, arrival of a line-control interrupt, activation of an operator interrupt, etc. Each of these events is tagged with the time of day, to the best resolution provided by the system (millisecond , microsecond, or nano-second), that the event is to occur. If the system is extremely complex or if the number of terminals or event type is large, it may also be advisable to develop an event generation program that can create a random sequence of events within the time tolerances specified.

Each of the generated events is placed on an on-line event file and the supervisor is modified at system initialization time to disable the actual events. Further, an event recognition routine is loaded with the supervisor. This routine sets an interval timer interrupt to occur at the specific time of each event on the event file. A standard set of benchmark programs can then be initiated to provide a multiprogramming batch mode of processing. At each interval timer interrupt, the causing event is read from the event file and the event recognition routine causes a linkage to be established to the proper interrupt handling routine.

An additional modification must be inserted in each interrupt handling routine to process the event from the linkage information provided by the event recognition routine. Further, if the interrupt handling routine masks out any other events, this information must be returned to the event recognition routine so that future events will be held pending until the mask is removed. When the event file is exhausted, the system should be notified to terminate validation operations and to notify the operator of test completion.

As indicated earlier, this technique is frequently used to validate real-time oriented systems. Consequently, it seems quite likely that the modifications to the supervisor described will already exist in a vendor's pre-production version of the system. When this is the case, this capability can be easily provided by the vendor for the validation sequence. When such a capability has not been developed by the vendor, the modifications to the operating system will be quite extensive. However, if the event sequence to be tested is considered critical to operational performance, such modification may still be justified.

It should also be noted that no capability to display the results of event occurrence and system reaction has been specified. In this area, it is recommended that the event logging for post-mortem analysis capability (see Subsection 3.2) be employed to validate the event processing sequence.

3.4 Interactive Test Control

This validation technique is based upon the concept that the individual performing system validation should be allowed to interact with the system to structure and record the results of selected activities of the system. Jerry Grochow has described a capability

3.4 (Continued)

provided to the MULTICS system designers which utilizes a PDP-8 computer to display various system statistics and to selectively modify the system during operation (see Reference 7 in the Bibliography). The capability described herein is based somewhat upon this concept though it does not necessarily entail the use of a separate processor.

A dedicated on-line console device is designated for the use of the system validator and a privileged partition is provided for the interactive test control program. This program should be designed to display various portions of the system supervisor area on the console device and to, upon command, modify selected system variables within the supervisor. All references to the system supervisor are symbolic to prevent inadvertent modification of actual core locations by the system validator.

In essence, the interactive test program would provide the validator with the following on-line commands:

1) Display logical system elements - This command will provide a structured display of the various logical elements comprising the system. These elements would consist of, but are not limited to, the following:
   - main storage allocation,
   - secondary storage allocation,
   - resource allocations,
   - current job mix,
   - dispatching queue,
   - scheduling queue.
2) Halt or proceed with validation run - This command will cause temporary suspension or resumption of system processing. Suspension would normally be invoked prior to displaying the various system elements or prior to modifying a system element. Resumption would cause the test sequence to continue.

3) Modify selected system control elements - This command will set selected system variables to a specified status. Then, the system validator can create overload conditions by restricting the amount of core or number of devices available or by reducing the number of entries permitted in scheduling, dispatching and I/O request queues. Further, he can modify resource availability to either include or exclude specific devices to satisfy his immediate test objectives.

4) Proceed until specific conditions arise - This command will permit the system to run uninterrupted until a specified condition occurs. Normally, the validator would issue this command to allow the system to create the necessary testing environment that he wishes to validate. The types of conditions that would be recognized are suggested by the following:
   - 'n' entries in a dispatching, scheduling, or I/O request queue,
   - specific job initiation or termination,
   - activation of a specific supervisor routine (e.g., a roll-out),
   - an elapsed time interval, etc.

5) Force event occurrence - This command will cause the invocation of the processing routine which handles a selected event. The types of events to be invoked would be:
   - the hardware error control routine,
   - the program error control routine,
   - an external interrupt,
   - a power failure, etc.

6) Invoke or release the event logging routine - This command will cause the event logging mechanism described in Subsection 3.2 to be initiated or terminated. Normally, the event logging routine, if available, would be dormant until the actual test environment is created (command 4) and then activated. By using the event logging capability, the validator is afforded a rather comprehensive post-mortem analysis of the system reaction to his structured test.

3.5 Programmed Test Control

This validation technique is quite similar to the previous technique except that the validator does not have on-line access to the system. Instead, all of the on-line commands are made available to a privileged executing program which will direct the activation of selected conditions and monitor the results of the system reaction to these conditions. All of the commands described in Subsection 3.4 would be available to the privileged program in addition to a more comprehensive display of supervisor status variables.

It is felt that this approach is more realistic for smaller systems or for those that do not provide an on-line console capability.

3.6 Conclusions

This Section has described four areas in which a system-assisted validation methodology can be employed to increase the information available from the operating system validation process and the amount of control the system validator can exercise over the process. The implementation of these recommendations should involve a considerable amount of design review to further ascertain the types of information that are relevant to system testing/validation objectives. These recommendations constitute only the first step in an attempt to increase the precision of the validation proceedings.

However, it is noteworthy that most systems, particularly the larger and more complex operating systems, utilize a large number of testing aids during the system debug cycle. The fact that these aids are normally unavailable to the system validator decreases the sophistication, and ultimately, the value of the validation process itself. Consequently, when it is known that extensive system debugging aids exist, it is strongly recommended that the system validator be made aware of these aids in order to design a more comprehensive program for ensuring that the proposed operating system fulfills the requirements of the intended application.

In this regard, the addition of a criterion to the system evaluation process whereby a vendor is also evaluated on the level and sophistication of his system testing aids is worth consideration. While testing/validation aids would not be a firm requirement, the award of "bonus points" for effective testing aids should encourage vendors to make available many of the routines which have already been developed and which would facilitate the design of more extensive and exhaustive system control tests.

SECTION IV
FUNCTIONAL TESTING REQUIREMENTS

4.1 Approach

The functional testing requirements are delineated by a tabulation within the integrated functional classification structure (see Reference 12 in the Bibliography). This method ensures a comprehensive listing of test requirements for test implementation selection.

Since different types of operating systems exhibit different operational characteristics, each functional requirement is related to a particular system type, viz., real-time, batch, time-sharing, etc. The criticality of each functional requirement is denoted by designating it as a fundamental or special case requirement. Finally, each requirement is referenced to a test package which is defined in Section 5.

This particular method of structuring operating system testing requirements provides a means by which test designers can procedurally relate any operating system to possible testing requirements.

4.2 Testing Requirements

Operating system testing requirements are presented within three functional areas:

Part I: Executive/Control Functions,

Part II: System Management Functions,

Part III: Data Manipulation Functions.

The testing requirements for each area are structured using a tabular format. For the Part I functions, this format consists of four columns entitled Functional Area, System Type, Capability Level, and Cross Reference to Test. The column entitled Functional Area contains a delineation of the functional areas within an operating system and the testing requirements found within each area. The column entitled System Type delineates the type of system within which each requirement occurs: RTS, Real-Time System; BPS, Batch Processing System; TSS, Time-Sharing System; ALL, all of these system types. The designation MPS in the System Type column indicates that the requirement is peculiar to multiprogramming systems. The column entitled Capability Level denotes whether a requirement is fundamental to a system type or occurs in special cases. The final column, entitled Cross Reference to Test, contains the alphabetic designation of one of the Test Packages presented in Section 5. For example, the letter "A" in the final column indicates that the recommended testing

of its associated criteria is included in "Test Package A - System Foundation" while the letter "C" in the final column indicates that the recommended testing of its associated criteria is included in "Test Package C - Normal Operation Control, etc. An example utilizing the system is as follows: It is necessary to validate Abnormal Termination which is functional area 1.1.5.3 on page 29. This function references test package "B". Turning to test package B, page 57, it is found that item "16" is Abnormal Termination Functions. Item "16" references Part I, Function 1.1.5.3, techniques (a)-(f). This reference is then found on page 74 and the validation technique stated is to "Force abnormal termination, and then observe the ensuing system action." Also, for certain requirements, a single asterisk or double asterisk is placed in the last column to denote the following:

* This system control function is somewhat unwieldy to test unless one of the techniques suggested in Section 3 is employed. Consequently, the given test should be replaced by a system-assisted technique, if available.

** This function is implicitly validated by one or more of the tests designed to verify other operating system functions. Consequently, a unique test validating this function is unnecessary.

For the Part II functions the tabular format consists of the columns Functional Area, Capability Level, and Cross Reference to Test. Each of these columns contain the same type of information as described for like columns occurring in the Part I tabulation.

For the Part III functions the tabular format consists of the columns Functional Area and Cross Reference to Test. Again, each of these columns contain the same type of information as described for like columns occurring in the Part I tabulation.

TESTING REQUIREMENTS - PART I: EXECUTIVE/CONTROL FUNCTIONS

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.0	JOB MANAGEMENT			
1.1	Job Control			
1.1.1	Scheduling			
1.1.1.1	Algorithmic Scheduling			
	recognition of job priorities	RTS,BPS	fundamental	C
	recognition of resources allocated/not allocated	BPS	fundamental	C*
	recognition of scheduling delay time	BPS	fundamental	C*
	recognition of job type (I/O, processor, etc.)	BPS	fundamental	C
	capability to modify job priorities by operator	BPS	special cases	C
	capability to modify job priorities by user	BPS	special cases	C
	capability to modify scheduling algorithm	BPS	special cases	C
1.1.1.2	Time Initiated Scheduling			
	recognition of time-of-day as a scheduling parameter	BPS	special cases	C*
	recognition of job deadline time as a scheduling parameter	BPS	special cases	C
	recognition of an elapsed time interval as a scheduling parameter	RTS,BPS	fundamental	C
1.1.1.3	Event Initiated Scheduling			
	recognition of specific events or interrupts	RTS,TSS	fundamental	C*
1.1.1.4	Program Initiated Scheduling			
	capability to initiate scheduling of symbionts	BPS,TSS	special cases	C
	capability to initiate scheduling of subprograms/subtasks	ALL	special cases	C

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.1.1.4	(cont'd.)			
	capability to provide scheduling for immediate execution	ALL	special cases	C
	capability to provide scheduling for asynchronous execution	MPS, ALL	special cases	C*
	capability to provide scheduling for subsequent execution	RTS, BPS	special cases	C
1.1.1.5	Conditional Scheduling			
	recognition of task completion/abnormal termination	BPS, RTS	fundamental	C
	recognition of internal switches set by prior task	BPS	special cases	C
	recognition of error code set by prior tasks/job steps	BPS	special cases	C
	recognition of externally set switches	BPS	special cases	C
	specification of conditional logic on job control cards	BPS	special cases	C
	capability to specify conditional scheduling at the job level	BPS	special cases	C
	capability to specify conditional scheduling at the job step level	BPS	special cases	C
	capability to specify conditional scheduling at the task level	BPS	special cases	C
1.1.1.6	Scheduling Queue Maintenance			
	capability to maintain scheduling queues	ALL	fundamental	C*
1.1.2	Resource Allocation			
1.1.2.1	Core Storage Allocation			
	capability to provide static (fixed) core allocation for:			
	program expansion,	BPS	fundamental	C**
	I/O buffers,	ALL	fundamental	C**

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.1.2.1	(cont'd.)			
	common areas,	BPS,RTS	fundamental	C**
	subtask execution.	ALL	fundamental	C**
	capability to provide dynamic core allocation for:			
	program expansion,	MPS,ALL	special cases	C*
	I/O buffers,	MPS,ALL	special cases	C*
	common areas,	MPS,ALL	special cases	C*
	subtask execution.	MPS,ALL	special cases	C*
	capability to provide dynamic core allocation through storage pools	MPS,ALL	special cases	C**
	capability permitting common (shared) core allocation between tasks of the same job	MPS,BPS	special cases	C
	capability providing storage protection against unauthorized program access	MPS,TSS	fundamental	C
	capability to provide storage protection against unauthorized I/O processor access	MPS,ALL	fundamental	C
	capability to provide storage write protection	MPS,ALL	fundamental	C
	capability to provide storage read protection	MPS,ALL	fundamental	C
1.1.2.2	I/O Device Allocation			
	capability to dynamically allocate devices/files	MPS,TSS	special cases	C
	capability to allocate actual physical devices	ALL	special cases	C
	capability to allocate devices according to access method	TSS,BPS	special cases	C
	capability to allocate devices according to device type	ALL	special cases	C

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.1.2.2	(cont'd.)			
	capability to allocate devices by symbolic references	ALL	fundamental	C
	capability to provide exclusive allocation of devices/files	ALL	fundamental	C
	capability to provide shared allocation of devices/files	MPS,TSS	fundamental	C
1.1.2.3	Common Subroutine Allocation			
	capability to support serially reusable subroutines	MPS,ALL	special cases	C
	capability to support reentrant subroutines	MPS,ALL	special cases	C
1.1.3	Program Loading			
	capability to load programs from the system library	ALL	fundamental	A**
	capability to load programs from a user library	BPS,TSS	special cases	A
	capability to load programs from the input stream	BPS,TSS	fundamental	A
	capability to load programs in relocatable form	ALL	fundamental	A
1.1.3.1	Structure Control			
	capability to support simple program structures	ALL	special cases	C**
	capability to support overlay program structures	ALL	special cases	C
1.1.3.2	Loading Control			
	capability to initiate loading via control cards	BPS	fundamental	A**
	capability to initiate loading via explicit program references	ALL	special cases	A
	capability to initiate loading via implicit program references	ALL	special cases	A
	capability to initiate compaction of fragmented core upon task termination	MPS	special case	C*
	capability to initiate compaction of fragmented core when dictated by priority requirements	MPS	special cases	C*

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.1.3.2	capability to initiate compaction of fragmented core when directed by the operator	MPS, BPS	special cases	C*
	capability to provide automatic overlay loading	ALL	special cases	C
	capability to provide directed overlay loading	ALL	special cases	C
1.1.3.3	Swapping Control			
	capability to provide roll-in/roll-out	RTS, BPS	special cases	C*
	capability to allow programs to time share core storage	TSS	fundamental	C**
1.1.4	Event Monitoring			
1.1.4.1	Dispatching Control			
	capability to provide fixed time-slice dispatching	MPS, TSS	special cases	C*
	capability to provide variable time-slice dispatching	MPS, TSS	special cases	C*
	capability to provide contention (priority) dispatching	MPS, TSS	special cases	C*
1.1.4.2	Event Synchronization			
	recognition of time intervals	ALL	fundamental	C*
	recognition of abnormal termination	ALL	fundamental	B**
	recognition of unsolicited key-ins	RTS, TSS	special cases	C
1.1.4.3	Interrupt Processing Control			
	recognition of interrupt priorities	ALL	fundamental	C*
	capability to mask interrupts	ALL	fundamental	C

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.1.4.4	Program Limit Monitoring			
	specification of limits for execution time	ALL	fundamental	C*
	specification of limits for number of input records	BPS,TSS	special cases	C
	specification of line limits for printed output	BPS,TSS	special cases	C
	specification of limits for punched card output	BPS,TSS	special cases	C
	specification of limits for output records	BPS,TSS	special cases	C
	specification of limits for main storage utilization	ALL	special cases	C
	specification of limits for secondary storage utilization	BPS,TSS	special cases	C
1.1.5	Program Termination Processing			
1.1.5.1	Resource Deallocation			
	capability to explicitly close files	ALL	fundamental	C
	capability to explicitly release I/O devices	ALL	fundamental	C
	capability to explicitly release core devices	ALL	fundamental	C
	capability to implicitly close files	ALL	special cases	C
	capability to implicitly release I/O devices	ALL	special cases	C*
	capability to implicitly release core devices	ALL	special cases	C*
1.1.5.2	Summary Information Outputting			
	capability to provide error summaries	ALL	fundamental	B
	capability to provide summaries of CPU time utilization	BPS,TSS	fundamental	C*
	capability to provide summaries of device utilization	BPS,TSS	special cases	C
	capability to provide summaries of file access statistics	ALL	special cases	C

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.1.5.3	Abnormal Termination			
	capability to dump core	ALL	fundamental	B**
	capability to dump files	ALL	fundamental	B**
	capability to execute a specified termination program	ALL	fundamental	B
	capability to initiate recovery procedures	ALL	special cases	B
	capability to notify the operator of abnormal terminations	ALL	fundamental	B
	capability to notify remote terminal users of abnormal terminations	BPS,TSS	fundamental	B
1.2	I/O Control			
1.2.1	I/O Scheduling			
	capability to queue I/O requests by channel	ALL	special cases	C
	capability to queue I/O requests by device	ALL	special cases	C
1.2.1.1	Device Resolution			
	capability to specify device assignment by input stream control cards	BPS,TSS	special cases	A
	capability to specify device assignments by operator commands	ALL	special cases	A
	capability to specify device assignments by program requests	ALL	special cases	A
	capability to specify device assignment by an interactive user	TSS	special cases	A
1.2.1.2	Request Stacking			
	capability to permit specification of device priority	ALL	special cases	C
	capability to permit specification of request priorities	RTS	fundamental	C

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.2.1.3	Alternate Routing Control			
	capability to initiate alternate channel/device selection automatically	ALL	fundamental	A
	capability to initiate alternate channel/device selection by the operator	ALL	special cases	A
1.2.2	Data Transfer			
1.2.2.1	Buffering Control			
	capability to provide system buffer pools	ALL	special cases	C
	capability allowing user buffer pools	ALL	special cases	C
	capability to provide exchange buffering	ALL	special cases	C
	capability to provide chained segment buffering	ALL	special cases	C
	capability to allow buffer assignment via job control statements	BPS,TSS	special cases	C
1.2.2.2	Data Code Translation			
	capability to convert data to/from device oriented coding schemes (e.g., paper tape formats)	ALL	fundamental	A
1.2.3	Device Manipulation			
	capability to permit forms control through specific requests	ALL	special cases	A
	capability to permit forms control via control characters embedded in output records	ALL	special cases	A
	capability to provide card stacking through direct commands	ALL	special cases	A

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.2.3	(cont'd.)			
	capability to permit card stacking through control characters imbedded in output records	ALL	special cases	A
	capability to position sequential access devices	ALL	fundamental	A
1.2.4	Remote Terminal Support			
	capability to communicate with the central computer interactively	TSS	fundamental	C
	capability to communicate with the central computer in the remote batch mode	BPS	special cases	C
	capability to provide concurrent remote terminal activity	MPS,TSS	special cases	C
	capability to provide inter-terminal communication	ALL	special cases	C
	capability to provide operator/remote terminal user communication	ALL	special cases	C
	capability to permit operator control over remote terminal activity	ALL	special cases	C
1.3	System Communication			
	capability to provide device independent communication formats	ALL	special cases	C
1.3.1	System Startup			
	capability to startup the entire system	ALL	fundamental	A
	capability to startup on a partition by partition basis	MPS,ALL	special cases	A
	capability to startup using catalogued procedures	ALL	special cases	A
	capability to respecify system generation parameters	ALL	special cases	A
	capability to specify device availability	ALL	fundamental	A

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.3.1	(cont'd.)			
	capability to permit controlled system reconfiguration	ALL	special cases	A
1.3.1.1	System Initialization			
	capability to modify partition sizes	MPS,ALL	special cases	A
	capability to modify/assign partition priorities	MPS,ALL	special cases	A
	capability to modify/assign time-slicing specifications	TSS,BPS	special cases	A*
	capability to schedule user initiation programs	ALL	special cases	A
	capability to request time/date specification	ALL	fundamental	A
1.3.1.2	System Restart			
	capability to employ user restart programs	RTS,BPS	fundamental	A
	capability to automatically restart jobs that were executing when the system halted	RTS BPS,TSS	fundamental special cases	A A
	capability to automatically reschedule queued jobs	ALL	special cases	A
	capability to reconfigure the system in the event of malfunction and maintain continuity of operation	BPS,TSS RTS	special cases fundamental	A A
1.3.2	Job Control Communication			
1.3.2.1	Non-Interactive Control			
	capability to permit job control from the operator console	ALL	fundamental	A
	capability to permit job control from remote terminals	BPS,RTS TSS	special cases fundamental	A A
	capability to use catalogued job control procedures	BPS,TSS	special cases	A
	capability to modify catalogued job control procedures	BPS,TSS	special cases	A

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.3.2.2	Interactive Job Control			
	capability to provide interactive job control through a local console	BPS,TSS	fundamental	A
	capability to provide interactive job control through a remote console	TSS, BPS	fundamental special cases	A A
1.3.3	Input/Output Stream Control			
	capability to provide automatic editing of job control command formats	BPS,TSS	fundamental	B
1.3.4	Resource Status Modification			
	capability permitting operator control of system resource status	ALL	fundamental	A
	capability to recognize the following device conditions: available, assigned, down, reserved, test mode.	ALL	fundamental	A*
	capability to permit the following types of resource modification: addition, deletion, replacement, switching.	ALL	fundamental	A
1.3.5	System Status Interrogation			
	capability to display the status of the system upon request	ALL	fundamental	C
	capability to display the status of the system continuously	ALL	fundamental	C**

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
1.4	Recovery Processing			
1.4.1	Checkpointing			
	capability to provide a checkpoint initiation by a program request	BPS, RTS TSS	fundamental special cases	B B
	capability to provide system-initiated checkpoints	BPS, RTS	special cases	B
	capability to provide checkpoint initiation by an operator request	ALL	fundamental	B
	capability to provide checkpoint initiation by an interactive user	TSS	special cases	B
	capability to permit user assignment of checkpoint files	ALL	special cases	B
	capability to provide automatic assignment of checkpoint files	ALL	fundamental	B
	capability to provide multiple checkpoint records	ALL	special cases	B
1.4.2	Restarting			
	capability to initiate a restart by a job control command	BPS, TSS	fundamental	B
	capability to initiate a restart by a user program request	ALL	special cases	B
	capability to initiate a restart by an operator request	ALL	fundamental	B
	capability to initiate a restart by an interactive terminal user	TSS, BPS	special cases	B
	capability to restart from a point other than the last one	ALL	special cases	B
	capability to restart from the beginning of a job step	BPS	special cases	B
	capability to provide automatic replacement of refreshable modules	ALL	special cases	B
	capability to reposition sequential input/output data files	ALL	fundamental	B

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
2.0	DIAGNOSTIC ERROR PROCESSING			
2.1	Hardware Error Control			
2.1.1	Error Correction			
	capability to detect CPU errors	ALL	fundamental	B*
	capability to detect I/O device errors	ALL	fundamental	B*
	capability to detect I/O channel or I/O processor errors	ALL	fundamental	B*
	capability to detect storage parity errors	ALL	fundamental	B*
	capability to detect co-processor errors	ALL	special cases	B*
	capability to detect power-failures	ALL	fundamental	B*
	capability to provide linkage to user routines upon error detection	ALL	special cases	B
	capability to provide alternate I/O routing	ALL	special cases	B
2.1.2	Error Notification			
	capability to provide operator console error messages	ALL	fundamental	B**
	capability to provide interactive user console error messages	TSS, BPS	fundamental	B**
	capability to permit subroutines and tasks to return error codes to calling programs	ALL	fundamental	B
	capability to update and maintain error statistics files	ALL	special cases	B
	capability to provide diagnostic logout of permanent errors	ALL	special cases	B
	capability to provide an error trace showing the events leading to an error	ALL	special cases	B

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
2.1.3	Error Recovery			
	capability to provide system reconfiguration via alternate device utilization	RTS BPS,TSS	fundamental special cases	B* B*
	capability to provide system reconfiguration via controlled system degradation	RTS BPS,TSS	fundamental special cases	B* B*
	capability to permit on-line diagnostic device testing	ALL	special cases	B
	capability to provide automatic restart from a system-maintained checkpoint	ALL	special cases	B*
2.2	Program Error Control			
2.2.1	Error Correction			
	capability to detect arithmetic errors	ALL	fundamental	B
	capability to detect invalid instructions	ALL	fundamental	B
	capability to detect privileged instructions	ALL	fundamental	B
	capability to detect invalid address errors	ALL	fundamental	B
	capability to detect storage protection errors	ALL	fundamental	B**
	capability to detect invalid data errors	ALL	fundamental	B
	capability to provide linkage to user routines upon detection of a program error	ALL	special cases	B
	capability to provide interactive correction procedures	RTS,TSS	special cases	B

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
2.2.2	Program Error Notification			
	capability to output program error notification on the operator's console	RTS BPS,TSS	fundamental special cases	B B
	capability to provide abnormal termination indicators	ALL	fundamental	B
	capability to permit job steps to set error indicators for subsequent job steps	BPS	special cases	B
	capability to provide error notification to interactive users	TSS	fundamental	B
2.2.3	Program Termination			
	capability to provide conditional termination when a specified error level is reached	ALL	special cases	B
	capability to initiate abnormal termination by an operator command	RTS,BPS	fundamental	B
	capability to initiate abnormal termination by a user program request	ALL	fundamental	B
	capability to initiate abnormal termination by an interactive user request	TSS	fundamental	B
2.3	Interface Error Control			
2.3.1	Operator Key-In Editing			
	capability to edit operator commands	ALL	fundamental	B
	capability to provide command rejection upon job control command error	ALL	fundamental	B
2.3.2	Job Control Command Editing			
	capability to issue a request for clarification by the operator for command errors	ALL	special cases	B**

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
2.3.2	(cont'd.)			
	capability to provide job termination upon job control command error	BPS, RTS	fundamental	B**
2.3.3	Remote Terminal Communication Editing			
	capability to provide message format editing	TSS	fundamental	B
		BPS	special cases	B
	capability to edit command structures	TSS	fundamental	B
		BPS	special cases	B
	capability to edit data	TSS, BPS	special cases	B
	capability to provide error notification via coded messages	TSS	fundamental	B
		BPS	special cases	B
	capability to provide error notification via free format messages	TSS, BPS	special cases	B
	capability to provide error notification in tutorial message form	TSS	special cases	B
2.3.4	Program to System Link Verification			
	capability to recognize errors in linkage sequences	ALL	fundamental	B
3.0	Processing Support			
3.1.1	Real Time Clock Service			
	capability to provide the current date	ALL	special cases	A
	capability to provide the time of day	ALL	special cases	A
	capability to provide date conversion of facilities	ALL	special cases	A
	capability to provide time format conversion facilities	ALL	special cases	A
	capability to provide facilities for task interruption at a specified real time	RTS	fundamental	A*

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
3.1.2	Interval Timer Service			
	capability to provide an interrupt at the completion of a specified time interval	ALL	fundamental	A
3.2	Testing/Debugging Service			
3.2.1	Storage Dump Control			
	capability to provide snapshot storage dumps	ALL	fundamental	A
	capability to provide postmortum storage dumps	ALL	fundamental	A
	capability to dump all available storage	ALL	fundamental	A
	capability to dump resident supervisor storage	ALL	special cases	A
	capability to dump user storage areas	ALL	fundamental	A
	capability to dump I/O storage areas	ALL	fundamental	A
	capability to dump common storage areas	ALL	fundamental	A
	capability to dump all storage between specified starting and ending locations	ALL	fundamental	A
	capability to provide conditional dump display facilities	ALL	special cases	A
	capability to initiate dumps via control statements	BPS	special cases	A
	capability to initiate dumps via operator key-ins	ALL	fundamental	A
	capability to initiate dumps via interactive user key-ins	TSS	special cases	A
3.2.2	Tracing Control			
	capability to provide data tracing	ALL	special cases	A
	capability to provide instruction tracing	ALL	special cases	A
	capability to provide logic tracing	ALL	special cases	A

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
3.2.2	(cont'd.)			
	capability to provide supervisor service request tracing	ALL	special cases	A
	capability to provide subroutine call tracing	ALL	special cases	A
	capability to initiate tracing from control statements	BPS	special cases	A
	capability to initiate tracing from operator key-ins	RTS	special cases	A
	capability to initiate tracing from an interactive key-in	TSS	special cases	A
3.2.3	System Test Mode Control			
	facilities to ignore I/O requests	RTS	fundamental	B
		BPS,TSS	special cases	B
	facilities to reroute I/O requests	RTS	fundamental	B
		BPS,TSS	special cases	B
	facilities to log I/O requests	ALL	special cases	B
	facilities to simulate I/O error conditions	RTS	fundamental	B
		BPS,TSS	special cases	B
	capability to allow the user to override abnormal abort conditions	ALL	special cases	B
	capability to allow the user to override subsequent job step cancellation	BPS	special cases	B
	capability allowing the insertion of breakpoints in programs	ALL	special cases	A
	capability allowing the user to start or restart a program at a specified address	ALL	special cases	A
	capability permitting memory searching/displaying	RTS,TSS	special cases	A
	capability permitting memory modification	ALL	special cases	A

	FUNCTIONAL AREA	SYSTEM TYPE	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
3.3	Logging and Accounting			
3.3.1	Maintaining Job Charge Information			
	capability to record and maintain job charge information	BPS,TSS	fundamental	C
	capability to provide linkage to user-supplied accounting routines	BPS,TSS	fundamental	C
	capability to display job charge information at a user's terminal	BPS,TSS	fundamental	C
	capability to display job charge information at a central site device	BPS,TSS	fundamental	C
3.3.2	Maintaining Error Statistics			
	capability to accumulate information for a hardware error summary	ALL	fundamental	B
	capability to accumulate information for a program error summary	BPS,TSS	special cases	B
	facilities for the analysis of error statistics	ALL	special cases	B
	capability to provide a list of file access violation attempts	ALL	fundamental	B
3.3.3	Maintaining System Utilization Statistics			
	capability to maintain a summary by user account	BPS,TSS	fundamental	C
	capability to maintain a summary of file accesses	BPS,TSS	special cases	C
	capability to maintain a summary of system service requests	ALL	special cases	C
3.4	Program Accessible System Description Maintenance			
	capability to maintain current system status information	ALL	fundamental	C
	capability to maintain current system description information	ALL	fundamental	C

TESTING REQUIREMENTS - PART II: SYSTEM MANAGEMENT FUNCTIONS

	FUNCTIONAL AREA	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
	These functions may appear in any system type; therefore, the System Type column is deleted from the presentation.		
1.0	OPERATING SYSTEM MANAGEMENT		
1.1	System Generation		
	capability to specify system configuration: CPU's, I/O controllers, peripheral devices, on-line console devices, memory size, etc.	fundamental	A
	capability to incorporate user-developed routines	fundamental	A
	capability to assign default values for operator commands	fundamental	A
	capability to assign default values for job control commands	fundamental	A
	capability to modify the scheduling or dispatching algorithms	special cases	A
1.2	System Maintenance		
	capability to permit on-line supervisor patching	special cases	E
	capability to allow definition of new command	special cases	E
	capability to rename job control commands	special cases	E
	capability to alter default value specifications	special cases	E
2.0	PROGRAM MAINTENANCE		
2.1	Library and Directory Maintenance		
	capability to dynamically catalog load modules	special cases	E
	capability to dynamically catalog task/procedure definitions	special cases	E
	capability to statically catalog load modules	special cases	E
	capability to statically catalog relocatable modules	special cases	E
	capability to statically catalog source modules	special cases	E
	capability to statically catalog macro routines	special cases	E

	FUNCTIONAL AREA	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
2.1	(cont'd.)		
	capability to statically catalog task/job procedures	special cases	E
	capability to copy system libraries	special cases	E
	capability to specify library space allocation	special cases	E
	capability to punch or list library elements	special cases	E
	capability to display library elements	special cases	E
2.2	Load Module Generation		
	capability to bind multiple modules into a single load module	fundamental	A
	capability to resolve inter-module instruction linkages	fundamental	A
	capability to resolve inter-module data field references	fundamental	A
	capability to resolve, alter or patch binder-generated code	special cases	A
	capability to scan the system library for unresolved references	special cases	A
3.0	COMPILER INTERFACES		
	capability to recognize compiler parameters on OS control cards	special cases	A
	capability to use system-maintained compiler communication tables	special cases	A
	capability to use non-standard input symbionts for processing specially formatted compiler output files	special cases	C
	capability to use compilation error codes as conditional scheduling parameters by subsequent job tasks	special cases	C
	capability to link to system sort/merge routines	special cases	D
	capability to link to system peripheral conversion routines	special cases	A
	capability to link to data management system routines	special cases	D

	FUNCTIONAL AREA	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
4.0	MANAGEMENT SUPPORT UTILITIES		
4.1	Peripheral Device Support		
	capability to perform surface analysis	special cases	B
	capability to provide automatic defective track replacement	special cases	B
	capability to provide track replacement upon operator command	special cases	B
	capability to overwrite disk/drum/core as a file purging routine	special cases	B
4.2	System Simulation Routines		
	capability to simulate I/O device activity	special cases	C
	capability to simulate real-time interrupts	special cases	C
	capability to simulate message transmission	special cases	C
	capability to simulate message receipt	special cases	C
4.3	System Measurement Routines		
	capability to provide job throughput times	special cases	C
	capability to provide file or device utilization figures	special cases	C
	capability to provide a visual display of current system utilization	special cases	C
	capability to provide a visual display of past system utilization	special cases	C
4.4	Stand-Alone Utilities[1]		
4.4.1	Status Display		
	capability to dump core storage	fundamental	A
	capability to dump file storage areas	special cases	A
	capability to dump the contents of machine registers	fundamental	A

[1] Note - These capabilities are independent programs rather than routines within the supervisor; consequently, they provide a supplementary (and redundant) capability when the supervisor is inadvertently destroyed.

	FUNCTIONAL AREA	CAPABILITY LEVEL	CROSS REFERENCE TO TEST
4.4.1	(cont'd.)		
	capability to dump diagnostic log-out areas	special cases	A
	capability to dump read-only storage	special cases	A
4.4.2	Recovery Support		
	capability to rebuild message queues	special cases	B
	capability to rebuild system processing queues	fundamental	B
	capability to reconstruct on-line file transactions	special cases	B
	capability to re-initiate suspended processing	fundamental	B
	capability to re-establish communication line links	special cases	B

TESTING REQUIREMENTS - PART III: DATA MANIPULATION FUNCTIONS

	FUNCTIONAL AREA	CROSS REFERENCE TO TEST
	These functions may appear within any system type; also whether they are fundamental or special cases is dependent upon the inherent characteristics of each unique system; therefore, the System Type and Capability Level columns are deleted from this presentation.	
1.0	DATA MANAGEMENT	
1.1	File Management	
1.1.1	File Recognition	
	capability to locate files using catalogued addresses	A
	capability to recognize system-assigned labels	A
	capability to recognize user-assigned labels	A
	capability to locate data using hierarchical levels of cataloging	A
1.1.2	File Access Control	
	capability to restrict access to a protected element[1]	C
	capability to specify read-only access	C
	capability to specify selective write access	C
	capability to provide concurrent file access to a number of users	C
1.1.3	Backup and Restoration	
	capability to provide automatic file restoration	C
	capability to provide operator-initiated file restoration	C
1.2	I/O Support Facilities	
1.2.1	Data Access Control	
	capability to provide non-queued data access	C

[1]A protected element may be a volume, a file, a logical record, a physical record or a data element.

	FUNCTIONAL AREA	CROSS REFERENCE TO TEST
1.2.1	(cont'd.)	
	capability to provide automatic read-ahead (queued) data access	C
1.2.1.2	Keyed/Indexed Access Control	
	capability to provide automatic key computation	C
	capability to provide automatic index maintenance	C
	capability to provide access via hardware keys	C
1.2.1.3	Random Access Control	
	capability to permit direct data access	C
1.2.1.4	Teleprocessing Access Control	
	capability to provide message time stamping	C
	capability to provide input/output message routing	C
	capability to provide input/output message queueing	C
	capability to provide priority message recognition	C
	capability to provide periodic polling of teleprocessing lines	C
1.2.2	Data Blocking/Deblocking Control	
	capability to permit blocking/deblocking of fixed length records	C
	capability to permit blocking/deblocking of variable length records	C
	capability to permit blocking/deblocking of records of undefined length	C
1.2.3	Label Processing	
	capability for automatically generating system labels upon opening a file	C
	capability for generating system labels upon closing a file	C

	FUNCTIONAL AREA	CROSS REFERENCE TO TEST
1.2.3	(cont'd.)	
	facility permitting generation of user labels upon opening a file	C
	facility permitting generation of user labels upon closing a file	C
	capability to automatically check system labels	C
	capability to check labels upon user request	C
1.3	Data Management System Facilities	
1.3.1	Control Specification	
	capability to provide specification of formats for:	D
	files,	
	reports,	
	input data,	
	retrieval queries.	
1.3.2	Data File Generation and Maintenance	
	capability to structure sequential files	D
	capability to structure hierarchical files	D
	capability to structure indexed files	D
	capability to structure ring files	D
	capability to structure list files	D
	capability to validate input data by an equal value comparison	D
	capability to validate input data range verification	D
	capability to validate input data masked comparison	D
	capability to sequence check input data	D

	FUNCTIONAL AREA	CROSS REFERENCE TO TEST
1.3.2	(cont'd.)	
	capability to automatically truncate input data	D
	capability to automatically pad input data	D
	capability to encode input data	D
	capability to decode input data	D
	capability to modify input data by a constant factor	D
	capability to recognize input termination by a standard (embedded) field	D
	capability to recognize input termination by a special control character	D
	capability to update files according to conditional (logical) criteria	D
	capability to automatically update subordinate files when the master file is modified	D
	capability to restructure files	D
	capability to perform intra-file merging	D
	capability to perform inter-file merging	D
	capability to perform interactive error correction procedures	D
	capability to perform error correction using pre-established procedures	D
1.3.3	Data Qualification and Retrieval	
	capability to permit pre-stored fixed logic queries	D
	capability to permit pre-stored modifiable logic queries	D
	capability to permit a cue/response mode of file interrogation	D
	capability allowing Boolean operators in retrieval queries	D

	FUNCTIONAL AREA	CROSS REFERENCE TO TEST
1.3.3	(cont'd.)	
	capability allowing quantitative operators in retrieval queries	D
	capability allowing statistical operators in retrieval queries	D
	capability allowing application-defined operators in retrieval queries	D
	capability allowing constants as operands in retrieval queries	D
	capability allowing data fields as operands in retrieval queries	D
	capability allowing arithmetic expressions as operands in retrieval queries	D
	capability to query a single file	D
	capability to query an inter-file logic search	D
1.3.4	Data Output	
	capability to output page header labels	D
	capability to output page trailer labels	D
	capability to output data labels	D
	capability to specify data positioning	D
	capability to right/left justify data	D
	capability to edit output data	D
	capability to decode data values	D
	capability to tally occurrences of specific data values	D
	capability to total specific data elements	D
	capability to provide pagination control	D

	FUNCTIONAL AREA	CROSS REFERENCE TO TEST
1.3.4	(cont'd.)	
	capability to produce multiple report copies	D
	capability to provide user structured reports	D
	capability to provide system structured reports	D
	capability to provide interactively defined reports	D
2.0	DATA HANDLING UTILITIES	
2.1	Data Handling Utilities	
	capability to provide unformatted display facilities	A
	capability to provide system specified display formats	A
	capability to provide user specified display formats	A
2.2	Peripheral Device Support	
2.2.1	Volume Positioning	
	capability to provide magnetic tape positioning:	A
	backspacing,	
	rewinding,	
	unloading,	
	erasing.	
	capability to provide direct access device positioning	A
2.2.2	Media Copy Facilities	
	capability to provide facilities for copying punched card data	A
	capability to provide facilities for copying magnetic tape data	A
	capability to provide facilities for copying paper tape data	A

	FUNCTIONAL AREA	CROSS REFERENCE TO TEST
2.2.2	(cont'd.)	
	capability to provide facilities for copying random access storage data	A
	capability to provide facilities for copying main storage data	A
	capability to provide format conversion during data copying	A
	capability to provide code conversion during data copying	A
2.2.3	Data Editing Facilities	
	capability providing single file scanning/editing	A
	capability providing file comparisons	A
	capability providing selective field comparisons	A
2.2.4	Test Data File Support	
	capability to support transaction files	A
	capability to support terminal message files	A
	capability to support history (trace) files	A
	capability to support input or data file generation	A
	capability to provide control message generation	A
	capability to provide output file generation	A
3.0	Sorting and Merging	
3.1	Sort Module Development	
	capability providing control card parameter specification	D
	capability to perform ascending/descending output sequence	D
	capability to support single/multiple sort control fields	D

	FUNCTIONAL AREA	CROSS REFERENCE TO TEST
3.1	(cont'd.)	
	capability to support single/mixed data field formats	D
	capability to recognize alphanumeric field keys	D
	capability to recognize binary field keys	D
	capability to recognize zoned/packed decimal field keys	D
	capability to recognize floating point field keys	D
	capability to recognize user-specified collating sequence	D
3.2	Sort Module Execution	
	capability to sort independent data files	D
	capability to sort records provided by an internal record address table	D
	capability to sort full data records	D
	capability to sort only data record tags	D
	capability to selectively reduce record size by field selection	D
	capability to output sorted data to external data files	D
	capability to construct an internal record address table of sorted data records	D
	capability to include user coding for label processing	D
	capability to include user coding for input record insertion/modification/deletion	D
	capability to include user coding for output record insertion/modification/deletion	D
	capability to include user coding for blocking/deblocking control	D
	capability to include user coding for error processing	D

SECTION V
TEST DESIGN

5.1 Test Packages

This Subsection defines a series of Test Packages for validating the requirements delineated in Subsection 4.2. These Test Packages were developed by subdividing System Validation into five major areas, viz., System Foundation, System Error Detection and Recovery, Normal Operation Control, Special Features, and System/Program Maintenance. Then the functional classification scheme was analyzed to relate facilities, i.e., groups of similar requirements, to each of these areas. This identified the Validation Techniques (given in Subsection 5.2) which each Test Package should contain. Finally, a logical sequence was determined for conducting the tests comprising each Package. These sequences are not unique, and may, of necessity, be varied for the validation of a particular operating system.

As noted above, each Test Package is designed to validate groups of similar requirements. It should be noted that the execution of each Package is dependent upon the proper performance of its predecessors and, although the successful execution of each Test Package will implicitly validate some of the facilities tested in succeeding Packages, it is still necessary to verify these requirements explicitly as delineated in the latter Packages.

Each Package is described by a list of its component tests, each of which is designed to validate a specific facility. Each list entry gives the name of the function providing the facility to be validated, the part of the functional classification scheme in which this function and the facility are contained, the number of the function and facility within this part of the classification scheme and the corresponding Validation Technique(s). By using these packages as a model, test designers can select and order tests for the validation testing of any commercially available operating systems.

Test Package A - System Foundation

Certain requirements must be validated at the beginning of system testing, since their proper execution is essential to the further validation of the system. The following ordered tests are recommended for the verification of System Foundation.

1. System Startup - Part I, Function 1.3.1, Techniques (a)-(f)
2. System Initialization - Part I, Function 1.3.1.1, Techniques (a)-(c)
3. System Generation - Part II, Function 1.1, Techniques (a)-(c)
4. Program Loading - Part I, Function 1.1.3, Techniques (a)-(d)
5. Loading Control - Part I, Function 1.1.3.2, Technique (a)
6. Load Module Generation - Part II, Function 2.2
7. Compiler Interfaces - Part II, Function 3.0
8. Storage Dump Control - Part I, Function 3.2.1, Techniques (a)-(b)
9. Tracing Control - Part I, Function 3.2.2, Techniques (a)-(e)
10. Status Display - Part II, Function 4.4.1, Technique (a)
11. Non-Interactive Control - Part I, Function 1.3.2.1, Techniques (a)-(d)
12. Interactive Control - Part I, Function 1.3.2.2, Techniques (a)-(b)
13. Resource Status Modification - Part I, Function 1.3.4, Techniques (a)-(c)
14. Data Code Translation - Part I, Function 1.2.2.2
15. Data Handling Utilities - Part III, Function 2.0
16. File Location Recognition - Part III, Function 1.1.1, Techniques (a)-(c)
17. Device Resolution - Part I, Function 1.2.1.1, Techniques (a)-(d)
18. Device Manipulation - Part I, Function 1.2.3, Techniques (a)-(e)
19. Alternate Routing Control - Part I, Function 1.2.1.3, Techniques (a)-(b)
20. Real-Time Clock Service - Part I, Function 3.1.1, Techniques (a)-(c)
21. Interval Timer Service- Part I, Function 3.1.2
22. System Restart - Part I, Function 1.3.1.2, Techniques (a)-(d)

Test Package B - System Error Detection and Recovery

After System Foundation has been validated, the system's error handling capabilities should be verified. The following ordered tests are recommended.

1. Hardware Error Control - Part I, Function 2.1, Techniques (a)-(e)
2. Error Notification - Part I, Function 2.1.2, Techniques (a)-(e)
3. Error Correction - Part I, Function 2.1.1, Techniques (a)-(c)
4. Error Recovery - Part I, Function 2.1.3, Techniques (a)-(c)
5. Program Error Notification - Part I, Function 2.2.2, Techniques (a)-(d)
6. Program Error Correction - Part I, Function 2.2.1, Techniques (a)-(d)
7. Program Termination - Part I, Function 2.2.3, Techniques (a)-(b)
8. I/O Stream Control (Editing) - Part I, Function 1.3.3
9. Operator Key-In Editing - Part I, Function 2.3.1, Techniques (a)-(b)
10. Control Command Editing - Part I, Function 2.3.2, Techniques (a)-(b)
11. Remote Terminal Communication Editing - Part I, Function 2.3.3, Techniques (a)-(c)
12. Program to System Link Verification - Part I, Function 2.3.4, Techniques (a)-(b)
13. I/O Simulation - Part I, Function 3.2.3.1, Techniques (a)-(d)
14. Abnormal Termination Recognition - Part I, Function 1.1.4.2, Technique (b)
15. Abnormal Termination Service - Part I, Function 3.2.3.2, Technique (c)
16. Abnormal Termination Functions - Part I, Function 1.1.5.3, Techniques (a)-(f)
17. Volume Maintenance - Part II, Function 4.1.2
18. File Purging - Part II, Function 4.1.2.2
19. Checkpointing/Restarting - Part I, Functions 1.4.1/1.4.2
20. Recovery Support - Part II, Function 4.4.2, Techniques (a)-(e)
21. Maintaining Error Statistics - Part I, Function 3.3.2, Techniques (a)-(c)
22. Summary Information Outputting - Part I, Function 1.1.5.2, Techniques (a)-(d)

Test Package C - Normal Operation Control

This testing phase validates system performance under nearly "true" operational conditions. The following tests are recommended in the order given.

1. Structure Control, Part I, Function 1.1.3.1, Techniques (a)-(b)
2. System Communication, Part I, Function 1.3
3. System Simulation, Part II, Function 4.2, Techniques (a)-(b)
4. Compiler Interfaces, Part II, Function 3.0
5. Algorithmic Scheduling, Part I, Function 1.1.1.1, Techniques (a)-(e)
6. Time Initiated Scheduling, Part I, Function 1.1.1.2, Techniques (a)-(d)
7. Event Initiated Scheduling, Part I, Function 1.1.1.3, Techniques (a)-(b)
8. Program Initiated Scheduling, Part I, Function 1.1.1.4, Techniques (a)-(c)
9. Conditional Scheduling, Part I, Function 1.1.1.5, Techniques (a)-(d)
10. Scheduling Queue Maintenance, Part I, Function 1.1.1.6, Techniques (a)-(b)
11. Dispatching Control, Part I, Function 1.1.4.1, Techniques (a)-(c)
12. Core Storage Allocation, Part I, Function 1.1.2.1, Techniques (a)-(f)
13. I/O Device Allocation, Part I, Function 1.1.2.2, Techniques (a)-(d)
14. I/O Scheduling, Part I, Function 1.2.1, Techniques (a)-(b)
15. Request Stacking, Part I, Function 1.2.1.2, Techniques (a)-(b)
16. Common Subroutine Allocation, Part I, Function 1.1.2.3, Techniques (a)-(b)
17. System Status Display, Part I, Function 1.3.5, Techniques (a)-(b)
18. Resource Deallocation, Part I, Function 1.1.5.1, Techniques (a)-(b)
19. Buffering Control, Part I, Function 1.2.2.1, Techniques (a)-(e)
20. Compaction of Fragmented Core, Part I, Function 1.1.3.2, Techniques (b)-(d)
21. Swapping Control, Part I, Function 1.1.3.3, Techniques (a)-(b)
22. Event Synchronization, Part I, Function 1.1.4.2, Techniques (a)-(c)
23. Interrupt Processing Control, Part I, Function 1.1.4.3, Techniques (a)-(b)
24. Label Generation, Part III, Function 1.2.3, Techniques (a)-(b)
25. File Security Control, Part III, Function 1.1.2.1, Techniques (a)-(b)
26. Read/Write Access Control, Part III, Function 1.1.2.2, Techniques (a)-(c)
27. Sequential Access Control, Part III, Function 1.2.1.1, Techniques (a)-(b)
28. Keyed/Indexed Access Control, Part III, Function 1.2.1.2, Techniques (a)-(b)
29. Random Access Control, Part III, Function 1.2.1.3
30. Teleprocessing Access Control, Part III, Function 1.2.1.4, Techniques (a)-(g)
31. Concurrent Access Control, Part III, Function 1.1.2.3, Techniques (a)-(b)
32. Backup and Restoration, Part III, Function 1.1.3, Techniques (a)-(e)
33. Data Blocking/Deblocking, Part III, Function 1.2.2, Techniques (a)-(h)
34. Remote Terminal Support, Part I, Function 1.2.4, Techniques (a)-(f)
35. System Measurement Routines, Part II, Function 4.3
36. Maintaining Job Charge Information, Part I, Function 3.3.1, Techniques (a)-(d)
37. Maintaining System Utilization Statistics, Part I, Function 3.3.3, Techniques (a)-(b)
38. Current System Status Interrogation, Part I, Function 3.4.1, Techniques (a)-(h)
39. System Definition Interrogation, Part I, Function 3.4.2
40. Summary Information Outputting, Part I, Function 1.1.5.2, Technique (a)-(d)
41. Program Limit Monitoring, Part I, Function 1.1.4.4, Techniques (a)-(g)

Test Package D - Special Features

The following ordered tests should be employed to validate the special features provided by the operating system.

1. Compiler Interfaces, Part II, Function 3.0
2. Control Specification, Part III, Function 1.3.1, Techniques (a)-(c)
3. Structure Definition, Part III, Function 1.3.2.1, Techniques (a)-(b)
4. Retrieval Mode Control, Part III, Function 1.3.3.1, Techniques (a)-(b)
5. Query Processing, Part III, Function 1.3.3.2, Techniques (a)-(d)
6. Data Record Selection, Part III, Function 1.3.3.3, Techniques (a)-(c)
7. Input Transaction Processing, Part III, Function 1.3.2.3, Techniques (a)-(c)
8. Logical Record Maintenance, Part III, Function 1.3.2.4, Techniques (a)-(c)
9. Interactive File Maintenance, Part III, Function 1.3.2.5, Techniques (a)-(b)
10. File Reorganization, Part III, Function 1.3.2.6, Techniques (a)-(c)
11. Data Error Procedures, Part III, Function 1.3.2.7, Techniques (a)-(b)
12. Data Output, Part III, Function 1.3.4, Techniques (a)-(f)
13. Sorting and Merging, Part III, Function 3.0

Test Package E - System/Program Maintenance

The maintenance requirements of the operating system should be validated using the following ordered tests.

1. System Maintenance, Part II, Function 1.2, Techniques (a)-(c)
2. Dynamic Cataloging, Part II, Function 2.1.1, Techniques (a)-(b)
3. Static Cataloging, Part II, Function 2.1.2, Techniques (a)-(e)
4. Utility Functions, Part II, Function 2.1.3, Techniques (a)-(e)

5.2 Validation Methods

This Subsection defines software validation tests for confirming the presence and proper performance of the various facilities afforded by contemporary operating systems. As in the presentation of the testing requirements, the facilities and their corresponding Validation Techniques are structured within the functional classification scheme.

The particular tests presented here should not be considered unique. On the other hand, considerable effort has been devoted to design tests which are simple in nature but yet complete. Effort has also been devoted to stating the Technique clearly and concisely. In the description of many of the tests, the Technique necessary for validation is presented and the actual verification required upon successful execution is nothing more than a predesigned message from the test program to the operator. For example, Test I 1.1.1.1 (c), which validates the "recognition of scheduling delay time" is stated as follows: "Include a job in the scheduled test series with a priority sufficiently low to ensure that the job will not be scheduled during an established time limit." When this job does execute, it should notify the test conductor. To avoid unnecessary repetition, the requirement for notification is not always explicitly stated. In tests for which the means of determining success or failure is not obvious or may be ambiguous, the method is given. Also, remarks concerning the order in which the tests should be conducted are quite limited since the previous Subsection presents a recommended grouping and sequence.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
PART I: EXECUTIVE/CONTROL		
1.0 JOB MANAGEMENT		
1.1 Job Control		
1.1.1 Scheduling		
1.1.1.1 Algorithmic Scheduling	(a) Recognition of job priorities	(a) 1) Schedule a series of varying priority jobs and observe the execution sequence. 2) Schedule two jobs with different priorities but the same resource requirements.
	(b) Recognition of resources allocated/ not allocated	(b) Schedule a high-priority job requiring all available resources.
	(c) Recognition of scheduling delay time	(c) Include a job in the scheduled test series with a priority sufficiently low to ensure that the job will not be scheduled during an established time limit.
	(d) Recognition of job type 1) CPU-bound 2) I/O-bound	(d) 1)-2) Include two jobs in this test phase. One of these jobs should contain a control parameter indicating that it is CPU-bound and the other a parameter denoting that it is I/O-bound. Observe whether the system shares resources among all the scheduled jobs or allocates resources for long periods of time to these particular jobs.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.1.1.1 Algorithmic Scheduling (Cont'd.)	(e) Operator modification of job priorities	(e) Attempt to alter the original priorities assigned to the jobs in this test phase via the operator console. Then rerun these jobs and observe their execution sequence.
1.1.1.2 Time Initiated Scheduling	(a) Time-of-day recognition	(a) Include a job in this test phase that is to be scheduled at a particular time of day.
	(b) Elapsed time interval recognition	(b) Include a job in this test phase that is to be scheduled after a particular time interval.
	(c) Periodic interval recognition	(c) Include a job in this test phase that is to be scheduled after each elapsement of a specified time interval.
	(d) Deadline recognition	(d) Include a job in this test phase which must be scheduled no later than a specified time.
1.1.1.3 Event Initiated Scheduling	(a) Recognition of: 1) Process control interrupts 2) Communication interrupts 3) I/O completion interrupts 4) Task completion interrupts 5) Error condition interrupts 6) Unsolicited key-in interrupts 7) Operator interrupts	(a) 1)-7) In each case, force the interrupt to be issued.

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.1.1.3	Event Initiated Scheduling (Cont'd.)	(b) Interrupt priority recognition	(b) Simultaneously, force a group of interrupts, each of which is assigned a unique level.
1.1.1.4	Program Initiated Scheduling	(a) Scheduling for immediate execution	(a) From an executing program, initiate a call for another program to be executed at once (suspending the calling program).
		(b) Scheduling for subsequent execution	(b) From an executing program, initate a call for another program to be executed when the calling program terminates.
		(c) Scheduling for asychronous execution	(c) From an executing program, initiate a call for another program to execute concurrently with the calling program.
1.1.1.5	Conditional Scheduling	Scheduling upon recognition of: a) Task completion/abnormal termination b) Internal switches set by prior task c) Error code set by prior task d) Externally set switches	(a)-(d) The ability of the system to continue operation during the execution of multiple test programs validates scheduling based upon task completion; the introduction of abnormal termination, internal switches, error codes, and external switches will validate system operation as specified.
1.1.1.6	Scheduling Queue Maintenance		Display the scheduling queue periodically and verify that its contents correspond to the sequence in which the test programs are actually being executed.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
1.1.1.6	Scheduling Queue Maintenance (Cont'd.)	(a) Operator modification	(a) Use operator commands to delete and add jobs to the scheduling queue. Then observe the execution sequence.
		(b) Overload handling capability	(b) Use operator command to add jobs to the scheduling queue in excess of its capacity and observe the resulting system action.
1.1.2	Resource Allocation		
1.1.2.1	Core Storage Allocation	(a) Static (fixed) core allocation for: 1) Program expansion 2) I/O buffers 3) Common areas 4) Task execution	(a) 1)-4) These capabilities are verified implicitly throughout System Validation and can be directly verified by designing test programs requiring these functions to be performed.
		(b) Dynamic core allocation for: 1) Program expansion 2) I/O buffers 3) Common areas 4) Task execution	(b) 1)-4) Use a test program to request additional core storage, indicate core provided, and release core. Similarly, issue a request to each I/O device allocatable, verify that each device has been allocated, and then read/write data to validate dynamic I/O buffer allocation. Then release each device. Task execution can be validated by requiring the program to request a task; then verify its execution. If common areas are allocatable by the system, use two programs, each requiring access to a commonly defined data

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.1.2.1 Core Storage Allocation (Cont'd.)		pool. Log each request and instance of successful use of the allocatable area.
	(c) Dynamic core allocation through storage pools	(c) Execute two programs that request additional core. Then dump both programs to verify that they received core from the same area of memory.
	(d) Allocation of common (shared) core between tasks of the same job	(d) Execute two tasks designed to share core. Allow each task to alter a portion of the common area and then dump this area.
	(e) Storage protection against unauthorized program access	(e) Execute a test program which attempts to access a protected area of memory. Verify notification to the operator or to the user of the unauthorized access attempt.
	(f) Storage read/write protection	(f) The same type of tests can be performed to validate these functions as were utilized to validate ¦ 1.1.1.2(e). For additional verification of the write protect facility, attempt to write to a known area; then dump this area to verify that it has not been altered.
1.1.2.2 I/O Device Allocation	(a) Dynamic device/file allocation	(a) This facility is verified during the validation of dynamic core allocation.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.1.2.2 I/O Device Allocation (Cont'd.)	(b) Allocation of the following: 1) Actual physical devices 2) Devices according to access method 3) Devices according to device type 4) Devices by symbolic reference	(b) 1)-4) This facility is verified implicitly throughout System Validation.
	(c) Exclusive allocation of devices/files	(c) Attempt to access a device/file previously assigned to another program. This can be done by physical device reference (if this method is supported). This facility can also be tested by using a program which requests all devices, and a second program that requests one, or several, devices. The failure of the second program to gain access to any device constitutes verification.
	(d) Shared allocation of devices/files	(d) Employ several test programs which require the same devices/files. In a time-sharing system, verify concurrent servicing of the tests by monitoring each terminal. In a multiprogramming batch system, execute test programs requiring the same devices/files. To verify that known information was written into these files, dump them following test completion. The sharing feature can be verified by requiring the test programs to log each instance in which they gain access to the devices/files.

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.1.2.3	Common Subroutine Allocation	(a) Support for serially reusable subroutines (b) Support for reentrant subroutines	(a)-(b) Require a test program to call and execute a common subroutine. Whether the routine is serially reusable or reentrant is dependent upon its design. To verify commonality of subroutine usage in a multiprogramming system, require two programs in operation to request the same subroutine; then, for each program, dump the area of core containing the call to the subroutine. If the call addressesare identical, then each program used the same subroutine.
1.1.3	Program Loading	Program loading: a) From the system library b) From a user library c) From an input stream d) In relocatable form	(a)-(d) In each case, attempt to utilize the facility. The proper execution of the loaded program validates the facility.
1.1.3.1	Structure Control	(a) Support for simple program structures (b) Support for overlay program structures	(a)-(b) The loading and execution of a single program within a defined core area validates support of simple program structures, while the loading and execution of a program that exceeds a defined core area validates the overlay function.
1.1.3.2	Loading Control	(a) Initiate loading via: 1) Control cards	(a) 1) Successful execution of a program loaded under card control implicitly validates this facility.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.1.3.2 Loading Control (Cont'd.)	2) Explicit program references	2) Require an executing test program to request the loading of a test subroutine or an overlay segment. Execution of the loaded element verifies the facility.
	3) Implicit program references	3) Require an executing test program to access a non-resident routine. Upon recognizing this access attempt, the system should load the routine and re-execute the call.
	(b) Compaction of fragmented core:	(b) To validate this facility, execute a mix of known jobs and perform the following tests:
	1) Upon job termination	1) Before any job reaches termination, record the starting addresses of all of the jobs. After a job has terminated, record the starting addresses of all of the remaining jobs. Verify compaction by comparison.
	2) When dictated by priority requirements	2) Schedule a high priority job with memory requirements that will necessitate the compaction of the currently executing jobs. If the high priority job goes into execution, compaction is validated.

FUNCTION		FACILITIES		VALIDATION TECHNIQUES
1.1.3.2	Loading Control (Cont'd.)	3)	When directed by the system operator	3) During the processing of this job mix, dump core. Then request compaction and dump core again. Compare the two dumps with a program designed to read the dump files.
		(c)	Automatic overlay loading	(c) Attempt to load and execute a test program with core requirements that exceed available memory.
		(d)	Directed overlay loading	(d) Using specified control parameters, structure a program into overlays and attempt to load and execute it.
1.1.3.3	Swapping Control	(a)	Roll-in/roll-out	(a) Require a foreground program to request core for expansion that exceeds the amount of foreground memory available. Verify that the system rolls out background programs (to release core) and that the foreground program continues executing. Also, verify the resumption of execution of programs in the background area upon normal termination of the foreground program.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
1.1.3.3	Swapping Control (Cont'd.)	(b) Program time-sharing of core storage	(b) Issue requests for service from several time-sharing terminals and attempt to execute a program from each. Note the return of output to each terminal according to the system's time-sharing scheme (round-robin, priority, etc.).
1.1.4	Event Monitoring		
1.1.4.1	Dispatching Control	(a) Fixed time-slice dispatching	(a)-(b) Execute several test programs that read the real-time clock each time they are dispatched and record the readings against the program name. Allow these programs to execute for a given period of time and then terminate each. Print the recorded output and verify that the clock readings recorded by each program represent the same fixed increment of time. This same test program can also be employed to verify the correct implementation of a variable time slice scheme.
		(b) Variable time-slice dispatching	
		(c) Contention (priority) dispatching	(c) Execute several test programs, assigning each a different priority. At the initiation of each program, log its name and the reading of the real-time clock. This will verify the time of initiation of each program for comparison with its priority.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.1.4.2 Event Synchronization	(a) Recognition of time intervals	(a) Execute a test program that requests a "wait" for a given time interval. The test should record the clock reading prior to the wait request and immediately upon regaining control.
	(b) Recognition of abnormal termination	(b) Force a test program to terminate abnormally. Verify system recognition by monitoring operator messages.
	(c) Recognition of unsolicited key-ins	(c) Initiate an unsolicited key-in from the operator's console or from a user terminal. Monitor the system's response.
1.1.4.3 Interrupt Processing Control	(a) Recognition of interrupt priorities	(a) The verification of this facility can usually be performed by introducing external stimuli to the system. If operator inputs are specified as a high level of interrupt, recognition of unsolicited key-ins will partially verify this function. In some systems end-of-tape, hardware, and system errors, etc. cause interrupts at a level that requires immediate attention. These types of functions can be simulated for validation. A frequently used method for priority interrupt verification is the use of a hardware simulator to mimic these interrupts. Proper handling of these interrupts can be verified by observation.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
1.1.4.3	Interrupt Processing (Cont'd)	(b) Masking of interrupts	(b) Execute a test program which enables/disables different sets of interrupt conditions and which operates in an idle loop between each masking operation. While each mask is in effect, attempt to generate the disabled interrupts and verify that they are unacknowledged.
1.1.4.4	Program Limit Monitoring	Specification of limits for: a) Execution time b) Number of input records c) Printed output d) Punched card output e) Output records f) Main storage utilization g) Secondary storage utilization	(a)-(g) Specify all limits permitted by the system for a test program which is expressly designed to violate each of them.
1.1.5	Program Termination Processing		
1.1.5.1	Resource Deallocation	(a) Capability to explicitly: 1) Close files 2) Release I/O devices 3) Release core devices	(a) 1)-3) Schedule two programs, each of which requires the same devices/files. The first program should use a device/file, release it, and then go into an idle loop. Verify that the second program goes into execution as each device/file is released or after all of the devices/files have been released.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
1.1.5.1	Resource Deallocation (Cont'd.)	(b) Capability to implicitly: 1) Close files 2) Release I/O devices 3) Release core devices	(b) 1)-3) These facilities can be verified by scheduling two programs which require the same device/file. At the termination of the first program, the second should be able to access the device/file used by the previous program.
1.1.5.2	Summary Information Outputting	Capability to produce summaries of: a) Error statistics b) CPU time utilization c) Device utilization d) File access statistics	(a)-(d) This facility can be verified by executing a program with known characteristics and then comparing these characteristics against the system-produced summaries.
1.1.5.3	Abnormal Termination	(a) Provide a core dump (b) Provide a file dump (c) Execute a specified termination program (d) Initiate recovery procedures (e) Notify the operator of abnormal terminations (f) Notify remote terminal users	(a)-(f) Force abnormal termination, and then observe the ensuing system actions.
1.2	I/O Control		
1.2.1	I/O Scheduling	(a) Queue I/O requests by channel (b) Queue I/O requests by device	(a)-(b) Use a test program to initiate a known sequence of I/O requests and then display or record the contents of the I/O queue. The use of privileged instructions or the privileged access mode will probably be required to obtain the information in the queue.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
1.2.1.1	Device Resolution	Specify device assignment by: a) Input stream control cards b) Operator commands c) Program requests d) An interactive user	(a)-(d) In each case, attempt to utilize the facility and observe the results.
1.2.1.2	Request Stacking	(a) Specification of device priorities (b) Specification of request priorities	(a)-(b) Schedule a known sequence of I/O actions and display or record the I/O queues. Also, issue a priority command or initiate a request for priority I/O action. Then display or record the I/O queues.
1.2.1.3	Alternate Routing Control	(a) Automatic initiation of alternate channel/device selection (b) Manual initiation of alternate channel/device selection	(a)-(b) During the execution of a test program which produces known output, disable a channel and/or the devices that the program requires. By comparison with the previous output, verify system selection of an alternate channel and/or devices for the program. Perform the alternate channel/device selection manually after the system notifies the operator of their "down" status.
1.2.2	Data Transfer		
1.2.2.1	Buffering Control	(a) Provide system buffer pools (b) Allow user buffer pools (c) Provide exchange buffering (d) Provide chained segment buffering	(a)-(e) Different systems provide various methods of buffering. The test to validate buffering capabilities should verify provision of the buffering control specified rather than the implementation of buffering.

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.2.2.1	Buffering Control (Cont'd.)	(e) Allow buffer assignment via job control statements	
1.2.2.2	Data Code Translation	Convert data to/from device-oriented coding schemes	Execute a test program which utilizes all of the system's code conversion routines.
1.2.3	Device Manipulation	(a) Permit forms control through specific requests	(a)-(e) In each case, attempt to utilize the facility and observe the results.
		(b) Permit forms control via control characters embedded in output records	
		(c) Provide card stacking through direct commands	
		(d) Permit card stacking through control characters	
		(e) Provide positioning of sequential devices	
1.2.4	Remote Terminal Support	(a) Interactive communication with the central computer	(a)-(f) In each case, attempt to utilize the facility and observe the results.
		(b) Remote batch mode communication with the central computer	
		(c) Concurrent remote terminal activity	
		(d) Inter-terminal communication	
		(e) Operator/remote terminal user communication	
		(f) Operator control of remote terminal activity	

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.3	System Communication	Provide device independent formats	Address different devices using the same call format and observe the results.
1.3.1	System Startup	(a) Capability to startup the entire system	(a)-(c) Attempt the actual system start-up.
		(b) Capability to startup on a partition-by-partition basis	
		(c) Capability to startup using catalogued procedures	
		(d) Respecification of system generation parameters	(d)-(f) In each case, attempt to make changes in the system by utilizing the facility. Verify the system's recognition of these changes by executing applicable test programs.
		(e) Specification of device availability	
		(f) Controlled system reconfiguration	
1.3.1.1	System Initialization	(a) Modification of partition sizes	(a)-(b) During system initialization, exercise each of the options afforded by the system. If partition sizes are modifiable, alter them, and then schedule background and foreground test programs which record or display their starting addresses. If partition priorities can be assigned, invoke this operation and then, using the previously described test programs, verify that the foreground programs reside in the high priority partitions and that the background programs reside in the low priority partitions.
		(b) Modification/assignment of partition priorities	

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
1.3.1.1	System Initialization (Cont'd.)	(c) Modification/assignment of time-slicing specifications	(c) Alter the time-slicing specifications and employ the same kind of programs described for Tests I 1.1.4.1 (a)-(b).
1.3.1.2	System Restart	(a) Schedule user restart programs	(a)-(c) Force a system halt while a known test program scenario is in operation. Upon restart, this scenario should continue operation, or be totally restarted, depending upon the type of capability provided by the system.
		(b) Automatically restart jobs in execution at system halt	
		(c) Automatically reschedule queued jobs	
		(d) Reconfigure the system in the event of malfunction and maintain continuity of operation	(d) Using the simulator described in Tests I 2.0, mimic a device malfunction during the execution of a test program. System operation should continue. Compare the program's output to that produced during an uninterrupted run.
1.3.2	Job Control Communication		
1.3.2.1	Non-Interactive Control	(a) Capability to exercise job control via the operator console	(a) This facility is tested implicitly throughout System Validation.
		(b) Capability to exercise job control via local/remote terminals	(b) Attempt to utilize the facility and observe the results.
		(c) Use of catalogued job control procedures	(c)-(d) Employ the job procedures afforded by the system. If the system permits modification of the procedures, alter them and, by observation, verify their implementation.
		(d) Capability to modify catalogued job control procedures	

FUNCTIONAL AREA		FACILITIES	VALIDATION TECHNIQUES
1.3.2.2	Interactive Job Control	(a) Capability to exercise job control via a local console (b) Capability to exercise job control via a remote console	(a)-(b) In each case, attempt to utilize the facility and observe the result.
1.3.3	Input/Output Stream Control	Automatic editing of job control command formats	Enter erroneous control command formats into the I/O stream and note the result.
1.3.4	Resource Status Modification	(a) Operator control of system resource status	(a) Alter the status of the system resource via the operator console and observe the results. This facility is easily verified for system peripherals but requires test programs and core dumps to verify the alteration of memory status.
		(b) Recognition of the following device conditions: 1) Available 2) Assigned 3) Down 4) Reserved	(b) 1)-4) These facilities are verified implicitly throughout System Validation.
		(c) Permit the following types of resource modifications: 1) Addition 2) Deletion 3) Replacement 4) Switching	(c) 1)-4) In each case, attempt to utilize the facility and observe the result.

	FUNCTIONAL AREA	FACILITIES	VALIDATION TECHNIQUES
1.3.5	System Status Interrogation	(a) Display system status upon request (b) Display system status continuously	(a)-(b) Execute a known operational scenario which has a predictable status. Monitor the requested status or the continuously displayed status and compare it with that of the scenario.
1.4 1.4.1/ 1.4.2	Recovery Processing Checkpointing/ Restarting		Because of their close relationship, this Validation Techniques for Checkpointing and Restarting are described jointly.
1.4.1.1/ 1.4.2.1	Program Initiated Checkpointing/ Restarting		Select a test program that is known to have executed correctly which produces its output on a printer file. Insert a request for a program checkpoint by a suspension request employing a user-specified device. Specify the program termination routine as the restart address. Execute and then attempt to restart this version of the program. Following program completion, compare its output to that produced by the original version of the program.
1.4.1.2/ 1.4.2.2	System Initiated Checkpointing/ Restarting	(a) Initiation of checkpointing to accomplish task suspension and roll-out	(a) Select two test modules that are known to execute correctly which produce their output on a printer file. The second program should require considerably more core that the first. Insert a unique identification message

FUNCTIONAL AREA	FACILITIES	VALIDATION TECHNIQUES
1.4.1.2/ (Cont'd.) 1.4.2.2		(a) (Cont'd.) into each program to be output upon program completion. Assign a low priority to the first program and a high priority to the second. Subsequently, initiate execution of the second. Outputting of the second program's identification message first, and production of output by each program identical to that produced previously, verifies the facility.
	(b) Initiation of checkpointing to provide protective error recovery	(b) Select a test program that is known to have executed correctly which produces its output on a printer file. Re-execute this program, taking periodic system checkpoints while it is running. When the program issues a request to output its data, invoke the simulator (described in the tests for I 2.1 Hardware Error Control) to mimic a hardware device error. Attempt to restart the program from the last checkpoint taken. Compare its output to that produced previously.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.4.1.3/ 1.4.2.3 Externally Initiated Checkpointing/Restarting	Initiation of checkpointing/restarting from: (a) A control card (b) The operator console (c) An interactive user terminal	Select a test program that is known to have executed correctly which produces its output on a printer file. Then, for each facility afforded by the system, checkpoint and attempt to restart the program. Compare the output to that produced previously.
1.4.1.4 Checkpoint Notification	Provision of direct checkpoint notification to: (a) The console operator (b) The job output stream (c) The system log	For the tests outlined above, verify checkpoint notification to the proper recipients. Also, verify the accuracy of the message formats.
1.4.2.4 Device Repositioning		Prior to attempting restarting for the tests outlined above, rewind all of the sequential tape files.

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
2.0	DIAGNOSTIC ERROR PROCESSING		
2.1	Hardware Error Control	Detection of the following errors: (a) CPU errors (b) I/O device errors (c) I/O channel or I/O processor errors (d) Storage parity errors (e) Co-processor errors	(a)-(e) Where possible, simulate these hardware error conditions via software (with supervisory coding, if necessary). The capability to detect a particular error is validated when the corresponding error correction, notification, and recovery procedures are verified.
2.1.1	Error Correction	(a) Retry capability	(a) For error conditions that permit retries, mimic continual failures (via the software simulator) and verify the transmission threshold.
		(b) Alternate I/O routing	(b) Issue requests to use the system's primary I/O devices and attempt to transmit known data patterns. Use the simulator to mimic I/O hardware errors. Dump the contents of the secondary I/O devices to verify their substitution for those devices which are "down".

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
2.1.1	(Cont'd.)	(c) Control of linkage to user routines	(c) Simulate the occurrence of those hardware errors for which user modules have been developed.
2.1.2	Error Notification	(a) Operator notification	(a) For error messages output on the operator's console, verify their correspondence to the errors mimicked and also verify the accuracy of the message formats.
		(b) Interactive user notification	(b) Perform Test I 2.1.2(a) using remote terminals instead of the operator console.
		(c) Subroutine/task error notification to the calling program	(c) The verification required for this facility is self-explanatory.
		(d) Maintenance of error statistics files	(d) Manually maintain a log of all "errors" which are successfully mimicked during the validation of I 2.1.1. Upon completing the testing of I 2.1.1, dump the error statistics file and compare it to the log.
		(e) Error tracing	(e) Verify the capability to output messages stating the success or failure of various system component as data is transmitted through them.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
2.1.3	Error Recovery	(a) System reconfiguration 1) Alternate device utilization	(a) 1) Issue requests to utilize specific devices and attempt to transmit known data patterns. Use the simulator to mimic hardware errors. Dump the contents of the appropriate backup devices to verify their substitution for those devices which are "down".
		2) System degradation	2) Use the simulator to mimic hardware errors when requests are issued for the use of certain devices which service on-line processing requirements. Then observe the system's ability to continue processing the most critical requirements, viz., high-priority interrupt conditions.
		(b) Manual reconfiguration 1) Resource respecification 2) System regeneration	(b) 1)-2) The verification required for these facilities is self-explanatory.
		(c) Automatic restarting from a system checkpoint	(c) This capability is verified during the validation of I 1.4.1.2 System Initiated Restarting.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
2.2	Program Error Control		
2.2.1	Error Correction	(a) Control of linkage to user routines upon the detection of: 1) Arithmetic errors i) Division by zero ii) Exponent over/underflow iii) Integer overflow	(a) 1) i)-iii)Force these errors by executing routines designed to ensure their occurrence. The subsequent execution of the appropriate user-supplied routines verifies this linkage facility.
		2) Invalid data	2) Verify the capabilities to read data from and write data to the various I/O device types in the system-provided representations (fixed point, floating point, etc.) formatted in the standard record sizes for each device type. Also verify such facilities as end-of-file recognition, etc.
		(b) Control of linkage to system routines upon the occurrence of: 1) Instruction errors i) Invalid instruction	(b) 1) i) Attempt to execute an instruction with an operation code bit configuration which is not defined in the machine's instruction repertoire.
		ii) Privileged instruction	ii) Attempt to execute instructions within an application routine which are reserved for supervisory mode use.

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
2.2.1	(Cont'd.)		
		(c) Invalid address errors	(c) Generate, through indexing, the address of a non-existent location in main memory. Then attempt to execute an instruction which references this "location".
		(d) Storage protection errors	(d) Attempt to execute an instruction which references a cell in a protected area of main memory.
2.2.2	Error Notification	(a) Outputting of program error messages on the operator's console	(a) If real-time programs are to be processed by the system, temporarily alter some of them to produce errors. Note the outputting of related messages on the operator's console
		(b) Providing abnormal termination indicators	(b) Verify the production of off-line error messages upon abnormal termination.
		(c) Capability to permit job steps to set error indicators for subsequent job steps	(c) If the system can process batch programs, verify the ability of routines to make logic decisions based upon the detection of error flags set by routines previously executed within the job.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
2.2.2	(Cont'd.)	(d) Capability to provide error notification to interactive terminal users	(d) Verify the production of error messages at remote terminals upon the execution of jobs with "built-in" errors.
2.2.3	Program Termination	(a) Conditional termination upon reaching a specified error level	(a) Attempt to compile a program which has been designed to contain errors in excess of the total number permitted by the compiler
		(b) Initiation of abnormal termination by: 1) The system	(b) 1) Verify abnormal termination upon a user-program attempt to employ a privileged instruction or to reference a protected area of memory
		2) The operator	2) Verify the capability to allow operator initiation of termination procedures upon notification of program errors.
		3) A batch program	3) Verify the capability of a batch program to initiate termination upon detecting "built-in" errors
		4) An interactive user program	4) Verify the capability of a time-sharing program to initiate termination upon detecting "built-in" errors.

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
2.3	Interface Error Control		
2.3.1	Operator Key-In Editing	(a) Assumed default options	(a) Verify the capability of the operating system to continue execution in accordance with pre-established parameters after requests are made for console input and no responses are supplied.
		(b) Error notification by: 1) Coded messages	(b) 1) Enter erroneous key-ins. Verify the production of the proper error codes.
		2) Free format messages	2) Enter erroneous key-ins. Verify the production of the proper error messages.
		3) Tutorial messages	3) Issue a request (via the operator's console) containing insufficient information for a system service. Verify the system's ability to "guide" the operator in formulating the correct request.
2.3.2	Control Command Editing	(a) Job termination	(a) Submit jobs to execute in the batch mode with control cards containing various types of errors. Verify the system's capability to terminate these jobs. Also, verify the proper

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
2.3.2	(Cont'd.)		correspondence of error messages produced to the control card errors.
		(b) Command rejection	(b) Submit jobs for execution from an interactive terminal which contain erroneous commands. Verify the system's capability to terminate these jobs.
2.3.3	Remote Terminal Communication Editing	(a) Capability to edit: 1) Message formats 2) Command structures 3) Data structures 4) Character structures (b) Error notification by: 1) Coded messages 2) Free format messages 3) Tutorial messages	(a)-(b) For each case, (a) 1)-4), attempt to enter erroneous forms and verify the system's capability to terminate the job and/or notify the remote user of the errors. The notification procedures (b) 1)-3) may be validated using the same methods recommended for verifying ¶ 2.3.1 (b) 1)-3).
		(c) Editing of user/terminal ID's	(c) Attempt to enter the system using ID's which have not been defined to the system
2.3.4	Program to System Link Verification	(a) Calling sequence (b) Parameter list validation	(a)-(b) Verify the ability to call system routines from user programs. Verify this capability for routines which require calling parameters as well as for those which do not.

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
3.0	PROCESSING SUPPORT		
3.1	Timing Service		
3.1.1	Real-Time Clock Service	(a) Capability to provide: 1) Current date 2) Time of day	(a) 1)-2) Request the current date and time of day and output them.
		(b) Date/time conversion	(b) Same as Test I 3.1.1(a) 1). (The conversion routines will automatically be called when the request is made.)
		(c) Task interruption until a specified time	(c) Design a test program to request suspension until a specified time. When the test program regains control, require it to verify that this reading is approximately the same as the specified restart time.
3.1.2	Interval Timer Service	Task suspension for a specified amount of time	Design a test program to obtain the current time of day, set the interval timer for a known interval of time, and then, upon interruption, read the time of day again. The difference between the readings should be approximately the same as the interval for which the timer was set.
3.2	Testing/Debugging Service		
3.2.1	Storage Dump Control	(a) Snapshot storage dumps (b) Postmortem storage dumps	(a)-(b) In each case, attempt to utilize the facility.

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
3.2.2	Tracing Control	Capability to provide: (a) Data tracing (b) Instruction tracing (c) Logic tracing (d) Supervisor service request tracing (e) Subroutine call tracing	(a)-(e) In each case, attempt to utilize the facility.
3.2.3	System Test Mode Control		
3.2.3.1	I/O Simulation	Simulation facilities for: (a) Ignoring I/O requests (b) Rerouting I/O requests (c) Logging I/O requests (d) Simulating I/O error conditions	(a)-(d) Logically disconnect all system I/O devices and then attempt to utilize each of these facilities.
3.2.3.2	Abnormal Termination Service	(a) Automatic storage dumps	(a)-(b) These facilities are verified by Tests I 1.1.5.3(a)-(b).
		(b) Automatic file dumps	
		(c) User override of normal abort conditions	(c) Attempt to utilize the facility.
3.2.3.3	Interactive Testing Service	(a) Insertion of breakpoints in programs (b) Starting or restarting a program at a specified address (c) Memory searching/displaying (d) Memory modification (e) Error notification and override capability	(a)-(e) In each case, attempt to utilize the facility.
3.3	Logging and Accounting		
3.3.1	Maintaining Job Charge Information	(a) Recording and maintaining the following job change information:	(a)-(b) No test module may be designed to explicitly validate

(Continued on next page.)

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
3.3.1	(Cont'd.)	1) CPU time utilization 2) I/O channel and device time utilization 3) I/O record units 4) Main storage utilization 5) Secondary storage utilization 6) Remote terminal utilization 7) Job termination conditions	these facilities since this would require use of the facilities themselves. However, if vendor-developed documentation is available for application routines supplied with the system, it will probably contain this data for some of these items. In this case, these routines should be executed and their statistics should be compared to those given by the manufacturer. (See Test I 3.3.3 (a).)
		(b) Linkage to user-supplied accounting routines	
		(c) Displaying job charge information on a central site device	(c)-(d) Verify the accuracy of the formats for the information produced by Tests I 3.3.1 (a)-(b).
		(d) Displaying job charge information at a user's terminal	
3.3.2	Maintaining Error Statistics	(a) Accumulation of information for a hardware error summary	(a) After completing Test I 2.1.1 (b), output the hardware error summary file.
		(b) Accumulation of information for a program error summary	(b) Verify the accuracy of the program error summaries after executing each of the routines described in Test i 2.2.1(a).

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
3.3.2	(Cont'd.)		
		(c) Maintaining a record of file access violation attempts	(c) Verify that a correct record of file access violation attempts is maintained for Tests III 1.1.2.2(a)-(b).
3.3.3	Maintaining System Utilization Statistics		
		(a) Maintenance of a summary by user account	(a) Assign dummy account numbers to each job used in Test I 3.3.1(a). The number of distinct accounts should be less than the total number of jobs. After verifying I 3.3.1(a), sum, for each account, the totals for all of the jobs designated to be charged to the account. Compare these figures to those maintained by the system for each account.
		(b) Maintenance of a summary of file accesses	(b) Following Tests III 1.1.2.3 (c) 1) and (d) 1), output the file access summary statistics.
3.4	Program Accessible System Description Maintenance		
3.4.1	Current System Status Interrogation	System status interrogation for the following information:	
		(a) Number of jobs (b) Number of interactive users (c) List of active terminals (d) Main storage allocation	(a)-(h) For a known job mix, interrogate the system for each of these data which are maintained.

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
3.4.1	(Cont'd.)	(e) Secondary storage allocation (f) Device allocation (g) Device status (h) Elapsed execution time	
3.4.2	System Definition Interrogation	Interrogation of the system for a description of its current hardware configuration.	Execute a test module which utilizes several different types of devices and interrogate the system for current component status.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
PART II: SYSTEM MANAGEMENT		
1.0 OPERATING SYSTEM MANAGEMENT		
1.1 System Generation	(a) Operating system configuration specifications 1) CPU's 2) I/O devices 3) Console devices	(a) 1)-3) Design the System Validation tests so that every device is used at some point in the scenario
	(b) Incorporation of user-developed routines	(b)-(c) If these capabilities are employed during system generation, they will be verified when the applicable functions are validated.
	(c) Specification of default values	
1.2 System Maintenance	(a) Dynamic Maintenance 1) System reconfiguration i) Augmented system operation ii) Component replacement iii) Degraded system operation	(a) 1)i) - iii) These facilities will be verified by Tests I 2.1.3 (a) 1)-2).
	2) On-line patching	2) Invoke a privileged instruction and attempt to alter an area of protected memory. Dump this area.
	(b) Off-line maintenance 1) System regeneration 2) Patching	(b) 1)-2) There is no requirement to test these facilities.
	(c) User-controlled maintenance 1) New command definition 2) Command renaming 3) Operand renaming 4) Default value alteration	(c) 1)-4) Attempt to use each facility afforded by the system.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
2.0	PROGRAM MAINTENANCE		
2.1	Library and Directory Maintenance		
2.1.1	Dynamic Cataloging	(a) Load modules	(a) Dynamically catalog one of the System Validation modules to the system load module library or to the user load module library. Execute the cataloged module by calling it from the library.
		(b) Task/procedure definitions	(b) Design a procedure which contains a set of control cards different from any of those provided in the system procedure library. Utilize the procedure to execute a test program and immediately thereafter attempt to catalog the control cards to the procedure library. Attempt to execute the test program again using a single control card to reference the procedure.
2.1.2	Static Cataloging	(a) Load modules (b) Relocatable modules (c) Source modules (d) Macro routines	(a)-(d) Write a source code instruction set which is different from any other instruction set in the macro library for that compiler language and attempt to catalog it. Write a source code program which uses this macro and catalog this program to the source code library. Compile the source code library element

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
2.1.2	Static Cataloging (Cont'd)		and catalog the result to the relocatable library. Execute the linkage editing function to retrieve the element from the relocatable library, build a load module and catalog it to the load module library. Execute the program from the load module library.
		(e) Task/job procedures	(e) Design a procedure and catalog it to the procedure library. Then execute the procedure by referencing it on the procedure library.
2.1.3	Utility Functions		For the following tests, it is suggested that the System Validation test programs be used as elements for verifying the library functions.
		(a) Library space allocation	(a) Create a library on a secondary storage device. This may be either a new library or an expansion of one of the system libraries. The library used may be either a source, relocatable, or load module library. For a more extensive test, the procedures can be repeated for each library.
		(b) Punching/Listing/Displaying	(b) Punch and display one of the test modules. List the entire test library for use in verification of Tests II 2.1.3 (c)-(e).

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
2.1.3 Utility Functions (Cont'd.)	(c) Copying	(c) Alter the name of a specific module on the test library.
	(d) Renaming	(d) Delete one or more of the modules from the test library and repack the remaining modules.
	(e) Repacking library space	(e) Copy the test library to a new library area.
		List the final, copied version of the test library to verify that these functions were executed correctly.

2.2 Load Module Generation

This function is largely confirmed by the actual system generation process and the compilation and execution of the various verification tests being applied against the system. Care should be taken to verify the binding capabilities of the system by a test program comprised of multiple relocatable elements. There are two facilities which, if available, should be verified explicitly: alteration of binder-generated code and system library scanning for unresolved references.

The ability to alter binder-generated code is directly verified by making known alterations to a test program. As an example, a test routine could include two entry points. Upon branching to the first, the message "CODE HAS NOT BEEN ALTERED SUCCESSFULLY" is typed. Then control is unconditionally returned to the main routine. Upon branching to the second entry point, the message "CODE HAS BEEN ALTERED SUCCESSFULLY" is typed and control is returned to the main program. Write a test program which branches only to the first entry point. After compiling this test program with the routine, attempt to alter the branch in the test program to jump to the second entry point. Verification is performed by executing the job.

The ability to scan the system library for unresolved references may be verified by inserting a call to a named library element in one of the test programs. Relinking the test program and executing it will verify the inclusion of the named relocatable library element.

3.0 COMPILER INTERFACES

The various support services provided to the system language compilers via interfaces to the operating systems should be validated for each compiler (ALGOL, FORTRAN, COBOL, JOVIAL, etc.). This may be done quite easily by a group of programs written in compiler language which utilize or "trigger" and then verify (usually through proper operation and printed output) the availability of the interfaced capabilities for each particular compiler. These capabilities may include one or more of the following:

1) executive routine support for
   a) compiler parameters on OS control cards,
   b) system-maintained compiler communication tables,
   c) program testing/debugging control;
2) libraries in support of
   a) source programs,
   b) macros,
   c) subroutines;
3) system utilities in support of compiler language programs such as
   a) sort/merge programs,
   b) peripheral conversion programs,
   c) data management system programs.

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
4.0	MANAGEMENT SUPPORT		
4.1	Peripheral Device Support		This function will be verified implicitly by performing the usual System Initialization procedures prior to System Generation.
4.1.1	Volume Preparation		
4.1.1.1	Record/Track Formatting		
4.1.1.2	Directory Creation		
4.1.1.3	Space Allocation		
4.1.2	Volume Maintenance		Usually, verification of these facilities is not possible during the check-out of a new system. However, if a damaged media is discovered during the validation of another functional area, then it may be used to validate some of these facilities.
4.1.2.1	Diagnostic Verification		
4.1.2.1.1	Surface Analysis		
4.1.2.1.2	Track Replacement		
4.1.2.2	File Purging		To test this function, which usually consists of a tape erase or core or disk/drum/data cell overwrite, purge a known portion of secondary storage and verify this by using a separate program to dump the purged files.
4.2	System Simulation Routines	Capability to simulate: (a) Real-time interrupts (b) Communication facilities	(a)-(b) If system simulation facilities are available, they should be used to validate the interrupt handling and teleprocessing support functions. This will also validate the utilization of these features.

	FUNCTION	FACILITIES	VALIDATION TECHNIQUES
4.3	System Measurement Routines		Analyze the validation tests to determine those for which the system-measurable statistics may be derived a priori. Following these tests, output the computed statistics and compare them with the expected values.
4.4	Stand-Alone Utilities		Select a convenient point during System Validation to force a failure from which the system cannot recover using normal diagnostic routines. Immediately prior to this action, dump the message queues and system processing queues.
4.4.1	Status Display	Capability to display: a) Core storage b) Hardware registers c) File storage d) Logout areas e) Read-only storage	(a)-(e) In each case, execute the status display routine.
4.4.2	Recovery Support	Capability to support the rebuilding of: a) Message queues b) System processing queues c) On-line transactions d) Suspended processing e) Communication line links	Initiate the pre-failure environment reconstruction routines to provide the following types of recovery support: a) Rebuild message queues b) Rebuild system processing queues c) Reconstruct on-line transactions d) Re-initiate suspended processing e) Re-establish communication line links

(Continued on the next page.)

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
4.4.2	(Cont'd.)		Then dump the message and system processing queues for comparison with the first queue dumps and attempt to resume on-line transactions and execution of the suspended processing.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
PART III: DATA MANIPULATION		
1.0 DATA MANAGEMENT		
1.1 File Management		
1.1.1 File Location Recognition	(a) Location of files using: 1) Catalogued addresses 2) Label recognition	(a) Attempt to locate files by: 1) Searching the master directory 2) An on-line sequential label comparison search
	(b) Location of data using hierarchical levels of cataloging	(b) Search files at each cataloging depth, including the maximum
	(c) Maintenance of separate catalogs	(c) Apply Tests III 1.1.1(a) 1) & b) to catalogs containing system-assigned file labels and user-assigned file labels
1.1.2 File Access Control		
1.1.2.1 File Security Control		Attempt to access a protected unit* by:
	(a) User ID protection	(a) A user ID other than one of those authorized by the file's creator
	(b) Password protection	(b) A password other than that authorized by the file's creator
1.1.2.2 Read/Write Access Control	(a) Read only access	(a) Attempt to write to each unit that is write protected

*A protected unit may be a volume, a file, a physical record, a logical record, or a data element.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.1.2.2 (Cont'd.)	(b) Read and selective write access	(b) Attempt to write to the restricted areas of each selectively write-protected unit
	(c) Unrestricted access	(c) 1) Attempt to read all of the data contained by each unrestricted unit. 2) Attempt to write to the entire area of each unrestricted unit.
1.1.2.3 Concurrent Access Control	(a) Single user read only access	(a) 1) Force concurrent read attempts for each applicable unit. 2) Force single write attempts for each applicable unit.
	(b) Single user read and write access	(b) 1) Force concurrent read attempts for each applicable unit. 2) Force concurrent write attempts for each applicable unit.
	(c) Multi-user read only access	(c) 1) Force concurrent read attempts for each applicable unit. 2) For single write attempts for each applicable unit.
	(d) Multi-user read/single user write access	(d) 1) Force concurrent read attempts for each applicable unit. 2) Force concurrent write attempts for each applicable unit.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
1.1.3	Backup and Restoration Capabilities	(a) Automatic restoration (b) Operator-initiated restoration (c) Restoration from transaction data files (d) Restoration from grandfather files (e) Restoration from checkpoint files	(a)-(e) Perform operations that damage selected routines and data in various files and then verify the restoration of their prior contents by the method chosen.
1.2	I/O Support Facilities		
1.2.1	Data Access Control		
1.2.1.1	Sequential Access Control	(a) Basic sequential access	(a) 1) Attempt to access specified magnetic tape records 2) Attempt to access consecutively stored records on disks, drums, etc.
		(b) Queued access	(b) Verify the automatic read-ahead capability for the secondary storage devices
1.2.1.2	Keyed/Indexed Access Control	(a) Keyed access	(a) Attempt to access records by: 1) Selected data fields 2) Hardware keys
		(b) Indexed access	(b) Attempt to access records by: 1) Searching the field value/ record address directory associated with the file 2) Utilizing all possible levels of indexing

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.2.1.3 Random Access Control	Direct access	Attempt to access non-consecutive records from disk, drum, etc; random access can usually be verified by testing the record identification/storage address algorithm
1.2.1.4 Teleprocessing Access Control	(a) Automatic message time stamping	(a)-(b) Send a series of teleprocessing messages and verify that the time of receipt has been appended to the message header block.
	(b) Optional message time stamping	
	(c) Input/output message priority	(c) Generate unique terminal-to-computer and computer-to-terminal messages with assigned priorities. Use a single program to produce a set of varying priority messages for a terminal and observe the order in which they are dispatched.
	(d) Input/output message routing	(d) Generate unique terminal-to-computer, computer-to-terminal and terminal-to-terminal messages with assigned headers and note the sites where they are received.
	(e) Input/output message queuing	(e) Generate unique terminal-to-computer and computer-to-terminal messages and note if the order in which these messages are stored in the I/O queues is in accordance with the queuing scheme (priority, time of initiation, etc.)

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.2.1.4 (Cont'd.)	(f) Periodic polling	(f) Send a unique message from each terminal and maintain a log of all messages as they are received at the main computer.
	(g) Request polling	(g) Initiate polling by the central computer 1) If all terminals are to be polled, attempt to send a unique message from each terminal when it is to be polled and maintain a log of all messages as they are received at the main computer. 2) If only selected terminals are to be polled, activate these terminals and again apply Test III 1.2.1.4(g) 1). Then activate terminals which are not to be polled, attempt to send unique messages from them and note any occurrence of their messages in the central computer log.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.2.2 Data Blocking/Deblocking Control	(a) Locate mode blocking for system-specified size groups of: 1) Fixed length records 2) Variable length records	(a) Attempt to access system-defined size blocks of known information. Perform this test on data collections composed of each allowable type of record: 1) Fixed length 2) Variable length Dump the accessed data.
	(b) Locate mode blocking for user-specified size groups of: 1) Fixed length records 2) Variable length records 3) Undefined length records	(b) Attempt to access user-defined size blocks of known information. Perform this test on data collections composed of each allowable type of record: 1) Fixed length 2) Variable length 3) Undefined length Dump the accessed data.
	(c) Locate mode deblocking of system-specified size blocks of: 1) Fixed length records 2) Variable length records	(c) Attempt to access selected records containing known information from blocks of system-defined size. Perform this test on blocks comprised of each allowable type of record: 1) Fixed length 2) Variable length Dump the accessed data.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.2.2 (Cont'd.)	(d) Locate mode deblocking of user-specified size blocks of: 1) Fixed length records 2) Variable length records 3) Undefined length records	(d) Attempt to access selected records containing known information from blocks of user-defined size. Perform this test on blocks composed of each allowable type of record: 1) Fixed length 2) Variable length 3) Undefined length Dump the accessed data.
	(e) Move mode blocking for system-specified size groups of: 1) Fixed length records 2) Variable length records	(e) Attempt to move known information as a unit (block) of system-specified size. Perform this test on groups of each allowable type of record: 1) Fixed length 2) Variable length Dump the contents of the destination storage area.
	(f) Move mode blocking for user-specified size groups of: 1) Fixed length records 2) Variable length records 3) Undefined length records	(f) Attempt to move known information as a unit (block) of user-specified size. Perform this test on groups of each allowable type of record: 1) Fixed length 2) Variable length 3) Undefined length Dump the contents of the destination storage area.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
1.2.2	(Cont'd.)	(g) Move mode deblocking of system-specified size blocks of:	(g) Attempt to move selected records of known information from system-specified size blocks. Perform this test on blocks composed of each allowable type of record:
		1) Fixed length records 2) Variable length records	1) Fixed length 2) Variable length Dump the contents of the destination storage area.
		(h) Move mode deblocking of user-specified size blocks of:	(h) Attempt to move selected records of known information from user-specified size blocks. Perform this test on blocks composed of each allowable type of record:
		1) Fixed length records 2) Variable length records 3) Undefined length records	1) Fixed length 2) Variable length 3) Undefined length Dump the contents of the destination storage area.
1.2.3	Label Processing	(a) System label generation	(a) Generate dummy files and check for the presence or absence of a system label after open and/or close operations. If a label is written, verify its contents.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.2.3 (Cont'd.)	(b) User label generation	(b) Generate dummy files and check for the presence or absence of a user label after open and/or close operations. If a label is written, verify its contents.

FUNCTION		FACILITIES	VALIDATION TECHNIQUES
1.3	Data Management System Facilities		
1.3.1	Control Specification	(a) Specification of formats for files, reports, input data, and retrieval queries (b) Locating file description tables (c) Derivation of control functions	(a)-(c) The capabilities of the functional control modules are verified when the various facilities of a specific data management system are validated.
1.3.2	Data File Generation and Maintenance		
1.3.2.1	Structure Definition	(a) Sequential, hierarchical, indexed, ring and list structures (b) Normal, hierarchical, and inverted indexing schemes	(a)-(b) Attempt to structure each allowable type of file. Dump each file and verify its format.
1.3.2.2	Space Allocation	Storage media supported: 1) Tape 2) Disk 3) Drum 4) Mass storage	Validation of the various capabilities afforded by a data management system verifies space allocation for its data base.

The capabilities provided by 1.3.3 Data Qualification and Retrieval for locating data by their names or indices must be validated prior to verification of the next 5 functional areas (1.3.2.3 - 1.3.2.7) as the testing of their facilities requires data element and record retrieval.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.3.2.3 Input Transaction Processing	(a) Input validation	(a) Verify production of the proper error messages upon attempting to enter the following into the data base:
	1) Range verification	1) i) Elements which are less than the minimum allowed values for designated variables ii) Elements which are greater than the maximum allowed values for designated variables
	2) Masked character comparison	2) Elements containing an erroneous character in each checked position
	3) Sequence checking	3) Elements sequenced in other than the established ordering scheme
	(b) Input alteration	(b) Attempt to enter the following into the data base and, if successful, retrieve the corresponding stored entries:
	1) Automatic truncation	1) Elements which exceed the maximum allowable length for particular variables
	2) Automatic padding	2) Elements with less than the maximum number of characters which require leading or trailing blanks or zeroes
	3) Decoding	3) Elements in pre-established abbreviated forms

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.3.2.3 (Cont'd.)	4) Encoding	4) Elements in their actual or full representation
	5) Constant factor modification	5) Elements which are arithmetic operands (addends, factors, etc.)
	(c) Input termination	(c) Attempt to halt input transaction processing by:
	1) Termination by an embedded field	1) A non-standard control field
	2) Termination by a special character	2) A character or character set other than the designated termination symbol
1.3.2.4 Logical Record Maintenance		Perform the following tests and then retrieve the records and elements to be altered:
	(a) Updating by logical querying	(a) Attempt to update files according to conditional criteria
	(b) Multi-record logic	(b) Attempt to update data by queries which are effected only after the successful comparison of test values to elements of more than one record
	(c) Automatic subordinate file updating	(c) Update records in a logical substructure of a file which consequently necessitate the updating of records at a lower substructure

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.3.2.5 Interactive File Maintenance	(a) Overriding of data values (b) Update logic capabilities	(a)-(b)The validation techniques for this functional area are identical to those for Logical Record Maintenance and File Reorganization except that control is exercised in the conversational mode.
1.3.2.6 File Reorganization	(a) File Restructuring	(a) 1) Attempt to add new data fields to and delete existing data fields from the file; dump the resulting file 2) Attempt to alter the size of existing data fields; dump the resulting file
	(b) Intra-file merging	(b) Attempt to merge records contained within the file; dump the resulting file
	(c) Inter-file merging	(c) Attempt to merge records contained within several files to form a single new file; dump the new file
1.3.2.7 Data Error Procedures	(a) Interactive correction	(a) Attempt to enter erroneous elements into the data base via the conversational mode. Upon error notification, attempt to correct these elements and then retrieve their stored values.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.3.2.7 (Cont'd.)	(b) Attempt to enter erroneous elements into the data base. Then retrieve the corresponding stored elements, noting instances of automatic correction	(b) Pre-established correction procedure
1.3.3 Data Qualification and Retrieval		
1.3.3.1 Retrieval Mode Control	(a) Interrogation by pre-stored queries	(a) Attempt to retrieve known data elements by executing:
	1) Fixed logic queries	1) Queries which contain all of their operands and operators when they are called from a system library
	2) Modifiable logic queries	2) Queries in which the user has varied the operands and/or operators from those of the original queries in the system library
	3) Parameterized queries	3) Queries in which the user has supplied some or all of the operands and/or operators to the skeletal forms of the queries called from the system library
	(b) Interrogation by interactive queries	(b) Attempt conversational retrieval of known data elements by:
	1) Cue-response queries	1) Participating in a question/answer dialogue with the system
	2) Prompting queries	2) Responding to a series of system-provided query formulation aids

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.3.3.2 Query Processing	(a) Boolean, quantitative, arithmetic, statistical, and application defined operators (b) Logical connecting operators (AND, OR, NOT) (c) Nesting of logical operators (d) Use of constants, data fields, interim query results, and arithmetic expressions as operands	(a)-(d) The set of queries used in the validation of Retrieval Mode Control should include all of these facilities which are available since the validation of Query Processing is concomitant with that for Retrieval Mode Control
1.3.3.3 Data Record Selection	(a) Single file searching (b) Multi-file searching (c) Inter-file searching	(a)-(c) The set of queries employed in the validation of Retrieval Mode Control should include those necessary for validating Data Record Selection.

The following facilities should be verified where applicable for each type of report that may be prepared, e.g., user structured, system structured, or interactively defined.

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.3.4 Data Output	(a) Labeling	
	1) Headers/trailers	(a) 1) Attempt to output specific labels in designated positions at the top and bottom of each page or frame of the report
	2) Data labels	2) Attempt to label specific output values
	(b) Data formatting capabilities	
	1) Horizontal/vertical positioning	(b) 1)-2) Attempt to output specific data elements to designated positions on the output media
	2) Right/left justification	
	(c) Data altering capabilities	
	1) Data editing	(c) 1) Attempt to afix algebraic and dollar signs to specific output values
	2) Decoding	2) i) Attempt to output stored abbreviated data values in their actual or full representation ii) Attempt to output stored data values with leading zeros suppressed
	(d) Accounting capabilities	
	1) Counting	(d) 1) Attempt to determine the total number of occurrences of specific values for given data elements

FUNCTION	FACILITIES	VALIDATION TECHNIQUES
1.3.4 (Cont'd.)	(d) 2) Totaling	2) Attempt to output the sum of the stored values of specific data elements
	(e) Pagination control	(e) Attempt to output specific data to a page or frame designated relative to the current page or frame.
	(f) Multiple copy capability	(f) Produce multiple copies, i.e., copies on different device types, of a single report

2.0 DATA HANDLING UTILITIES

The explicit and individual verification of each type of peripheral device support facility would require an almost infinite battery of tests. The procedure outlined below defines a typical series of tests for validation of this functional area.

Step 1 Generate and store in a main memory work area several data sets. Each set should contain all of the characters representable in the system ordered according to the system collating sequence.

Step 2 Fill a second main memory work area with "nines" and write this area to magnetic tape followed by an end-of-file mark, thus creating a "dummy" tape file.

Step 3 Punch the test data (the contents of the first work area) into cards and paper tape, write it beyond the dummy file on magnetic tape, and to a specific area of a disk.

Step 4 Read the cards into a third main memory work area and compare its contents to the first work area, advising the test conductor via a console output device of any discrepancies. Similarly, verify the paper tape, magnetic tape, and disk copies of the test data.

Step 5 Dump the data in the first work area to the system printer using an unformatted display routine.

Step 6 Output the first work area to a CRT using a routine containing user format specifications.

Step 7 Apply field insertion and code conversion to the data in the first work area. Then output the area to the system printer.

3.0 SORTING AND MERGING

As in the case of DATA HANDLING UTILITIES the direct, isolated verification of each sort/merge facility would involve a very extensive battery of tests. Again, a compact scenario of representative tests will provide sufficient validation of this functional area. The major capabilities of sorting and related user-supplied routines are given in the list below. Combinations of these capabilities, selected to typify the actual verification requirements, are described in the subsequent tests.

Sort/Merge Options

I Input Source

- a) External Data Files
- b) Internal Record Address Table

II Data Format

- a) Integer
- b) Floating Point
- c) Zoned/Packed Decimal
- d) Alphanumeric
- e) Other

III Control Fields

- a) Single
- b) Multiple

IV Source of Control Parameters

- a) Control Cards
- b) Internal Linkage Parameters

V Output Collating Sequence

- a) Ascending
- b) Descending
- c) User Specified

VI Special Outputting Facilities

Data Conversion

VII User-Provided Output Facilities

- a) Input Record Insertion
- b) Input Record Deletion
- c) Input Record Modification
- d) Output Record Insertion
- e) Output Record Deletion
- f) Output Record Modification

VIII Output Destination

a) External Data Files
b) Internal Record Address Table

IX Output File Format

a) Blocked, Fixed-Length Records
b) Blocked, Variable-Length Records
c) Unblocked, Fixed-Length Records
d) Unblocked, Variable-Length Records
e) Variable - Blocked, Fixed-Length Records
f) Variable - Blocked, Variable-Length Records

Sort/Merge Validation Techniques

Tests III 3.0 (a) and III 3.0 (b) validate sorting capabilities while Test III 3.0(c) validates sorting and merge facilities. The test data files should be program generated. The success or failure of each test should be determined by a program that reads and processes the resulting sorted output file and verifies the correct record sequence.

Test III 3.0 (a)1) Create a data set composed of the alphabetic sequences A...A, AA...AB, ... , AA...AZ, AA...ABB, ..., AA...AZZ, ..., AB...B, ..., AZ...Z, where each member (sequence) of the data set is 26 characters in length. Then create the data set composed of the sequences B...B, BB...BCb, ..., BB...BZb, BB...BCCb, ..., BB...ZZb, ..., BC...Cb, ..., BZ...Zb, where each member of the set is again 26 characters in length and the last character b of each sequence except the first is a blank. Continue generating data sets until the algorithm is exhausted. Z...Z is the sole member of the last, i.e., the 26th, data set.

(2) Now sequentially number the elements of each set. The first set of sequences will be numbered 1 to 626; the second, 1 to 577; etc.

(3) Employ a random number generator (either vendor or user-developed) to create random integers. Attach a random number to each of these alphabetic-numeric combinations. Then write the data to a secondary storage unit using system I/O routines, if available.

(4) Call the sort routine and submit a control card indicating that the test data are to be sorted into ascending order using the generated set of random numbers. Submit another card defining the output blocking factor and record length, and indicating that the output is to be written in blocked fixed-length records.

(5) Submit subsequent control cards indicating that these two-field strings are to be sorted by their assigned sequence numbers, and that each alphabetic group for each sequence number is then to be sorted according to the system collating sequence and written to tape in unblocked fixed-length records.

(6) Following verification of (e), attempt output record modification by changing the first character of each sequence to a special character, e.g., an & (or +).

Test III 3.0(b) Again use the random number generator to create a set of random integers. Prior to writing this data to tape, convert the integers to floating point and divide each by three to create mixed floating-point numbers. After creating a multi-tape file, submit control cards indicating that the data are to be sorted into descending sequence. Output the file in unblocked variable-length records.

Test III 3.0(c)1) Recreate the file of data sets described in Test III 3.0 (a) 1) and write them to a secondary storage device.

2) Now sort each sequence type according to the collating sequence of the system so that all sequences of the form A...A, ..., Z...Z, are ordered into a group; all sequences of the form AA...AB, ..., BB...BCb, are ordered into a group; etc.

3) After all possible groups (626 total) have been created according to this pattern, attempt to merge them to form the original test data file.

APPENDIX I

BIBLIOGRAPHY

1. Appleman, Robert C., Generalized Procedures for System Testing and Evaluation of Time-Sharing and Multiprocessing Computer Systems, Vol. I: Entering the Third Generation, (TM-BN-001/001/02); System Development Corp.; Santa Monica, Cal.; 9 Aug. 1968.

2. Arden, B., & Boettner, D., Measurement and Performance of a Multiprogramming System; publication of the Second ACM Symposium On Operating Systems Principles; Princeton, New Jersey; p. 130, Oct. 20-22, 1969.

3. Balzer, R. M., EXDAMS - EXtendable Debugging and Monitoring System; AFIPS Conference Proceedings, AFIPS Press; Montvale, New Jersey; Vol. 34, p. 567; 1969.

4. Bryan, G. E.,& Shemer, J. E., The UTS Time-Sharing System: Performance Analysis and Instrumentation; publication of the Second ACM Symposium On Operating Systems Principles; Princeton, New Jersey; p. 147; Oct. 20-22, 1969.

5. Campbell, D. J.,& Heffner, W. J., Measurement and Analysis of Large Operating Systems During System Development; AFIPS Conference Proceedings (1968 Fall Joint Computer Conference); p. 903.

6. Cantrell, H. N.,& Ellison, A. L., Multiprogramming System Performance Measurement and Analysis; AFIPS Conference Proceedings; Thompson Book Co.; Washington, D. C.; Vol. 32, p. 213; 1968.

7. Grochow, J. M., The Graphic Display as an Aid in the Monitoring of a Time-Shared Computer System, (MAC-TR-54), 1968.

8. Hart, L. E., The User's Guide to Evaluation Products; Datamation, p. 32, Dec. 15, 1970.

9. Karush, Arnold D., Two Approaches for Measuring the Performance of Time-Sharing Systems; publication of the Second ACM Symposium On Operating Systems Principles; Princeton, New Jersey; p. 159; Oct. 20-22, 1969.

10. Kolence, Ken W., System Improvement by System Measurement; Data Base; Vol. 1, No. 4, p. 6; Winter 1969.

11. Martin, James, Programming Real-Time Computer Systems; Prentice-Hall, Englewood Cliffs, New Jersey; 1965.

12. McIntosh, Clinton S., et. al., Analysis of Major Computer Operating Systems (ESD-TR-70-377), for HQ ESD (AFSC), L. G. Hanscom Field, Bedford, Massachusetts, by The COMTRE Corp.; Coral Gables, Florida; August 1970.

13. Pinkerton, Tad B., Performance Monitoring in a Time-Sharing System; Communications of the ACM, Vol. 12, Number 11, p. 608; 1969.

14. Presser, L., & Melkanoff, M. A., Software Measurements and Their Influence Upon Machine Language Design; AFIPS Conference Proceedings; AFIPS Press; Montvale, New Jersey; Vol. 34, p. 733; 1969.

15. Saltzer, Jerome H., & Gintell, John W., The Instrumentation of MULTICS; publication of the Second ACM Symposium On Operating Systems Principles; Princeton, New Jersey; p. 167; Oct. 20-22, 1969.

16. Trapnell, F. M., A Systematic Approach to the Development of System Programs; AFIPS Conference Proceedings; AFIPS Press; Montvale, New Jersey; Vol. 34, p. 411; 1969.

17. Wulf, William A., Performance Monitors for Multi-Programming Systems; publication of the Second ACM Symposium On Operating Systems Principles; Princeton, New Jersey; p. 175; Oct. 20-22, 1969.

18. CIMS/I (Computer Installation Management System); Booth Resources International, Inc.; Los Angeles, Cal.

19. CIMS/II (Computer Installation Management System/Version II); Booth Resources International, Inc.; Los Angeles, Cal.

20. COMPUMETER II: Computing Efficiency, Inc.; Bohemia, New York.

21. DOS MURS (Machine Utilization Reporting System); Webster Computer Corporation; Danbury, Connecticut.

22. System Measurement Software: SMS/70 TDOS Problem Program Evaluation (SPPE, Version 1) Product Description; Boole & Babbage, Inc.; Palo Alto, Calif.; May 1970.

23. System Measurement Software: SMS/360 Configuration Utilization Efficiency (CUE, Version 2) Product Description; Boole & Babbage, Inc.; Palo Alto, Calif.; Rev. 1, June 1970.

24. System Measurement Software: SMS/360 Configuration Utilization Evaluator (MCUE, Version 1) Product Description; Boole & Babbage, Inc.; Palo Alto, Calif.; April 1970.

25. System Measurement Software: SMS/360 Data Set Optimizer (DSO, Version 1) Product Description; Boole & Babbage, Inc.; Palo Alto, Calif.; Rev. 1, June 1970.

26. System Measurement Software: SMS/360 Problem Program Evaluator (PPE, Version 2), (MPPE, Version 1) Product Description; Boole & Babbage, Inc.; Palo Alto, Calif.; April 1970.

APPENDIX II

RECORD OF TESTING AND MEASUREMENT INTERVIEWS

1.0 Purpose - General meeting on technical aspects of OS validation

Participants - Representatives of USAF (Hq. ESD); Project MAC; Honeywell, Inc.; Bolt, Beranek, and Newman; and The COMTRE Corporation

Date - 6 July 1970

2.0 Purpose - Discuss the validation aspects of operating systems as they relate to system procurement

Participants - Representatives of ESD/ESMDA, ESD/ESMCT, and The COMTRE Corporation

Date - 6 July 1970

3.0 Purpose - Discuss OS validation as it relates to maintaining the Burrough's B3500 Master Control Program

Participants - Representatives of ESD/ESMDA, AFDSDC/DIA, and The COMTRE Corporation

Date - 7 July 1970

4.0 Purpose - Discuss OS validation and the validation activities of the quality control section of DSDC

Participants - Representatives of ESD/ESMDA, AFDSDC/SCCQ, and The COMTRE Corporation

Date - 7 July 1970

5.0 Purpose - Discuss OS validation and the validation activities at DIA

Participants - Representatives of DIA/DIAMS-3 and The COMTRE Corporation

Date - 8 July 1970

6.0 Purpose - Discuss OS validation and the validation of the GECOS III system at the center

Participants - Representatives of AF/Data Services Center and The COMTRE Corporation

Date - 9 July 1970

7.0 Purpose - Discuss OS validation as it relates to the activities of the NMCSSC

Participants - Representatives of NMCSSC and The COMTRE Corporation

Date - 9 July 1970

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
The COMTRE Corporation
151 Sevilla Avenue
Coral Gables, Florida 33134

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
N/A

3. REPORT TITLE

OPERATING SYSTEM VALIDATION TESTING

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
None

5. AUTHOR(S) *(First name, middle initial, last name)*
William C. Mittwede
Kenneth P. Choate

6. REPORT DATE
January 1971

7a. TOTAL NO. OF PAGES
130

7b. NO. OF REFS
3 plus Bibliography

8a. CONTRACT OR GRANT NO.
F19628-70-C-0258

b. PROJECT NO.
6917

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
ESD-TR-71-81

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Deputy for Command and Management Systems,
Hq Electronic Systems Division (AFSC),
L G Hanscom Field, Bedford, Mass. 01730

13. ABSTRACT

This report presents functional testing requirements for use in the validation testing of computer operating systems. The requirements are structured in a tabular format and are applicable to the executive/control functions, system management functions and data manipulation functions of current commercially available operating systems. In concert with the tabulation of requirements for each of the operating system functions, further tabulation has also been performed relating the test requirements to the type of environment that the operating system must support: batch, real-time, or time-sharing. Basic testing procedures have been defined to verify the requirements and these testing methods have then been grouped into test packages.

DD FORM 1 NOV 65 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
validation testing operating system (computer) executive/control system management data manipulation environment requirements						